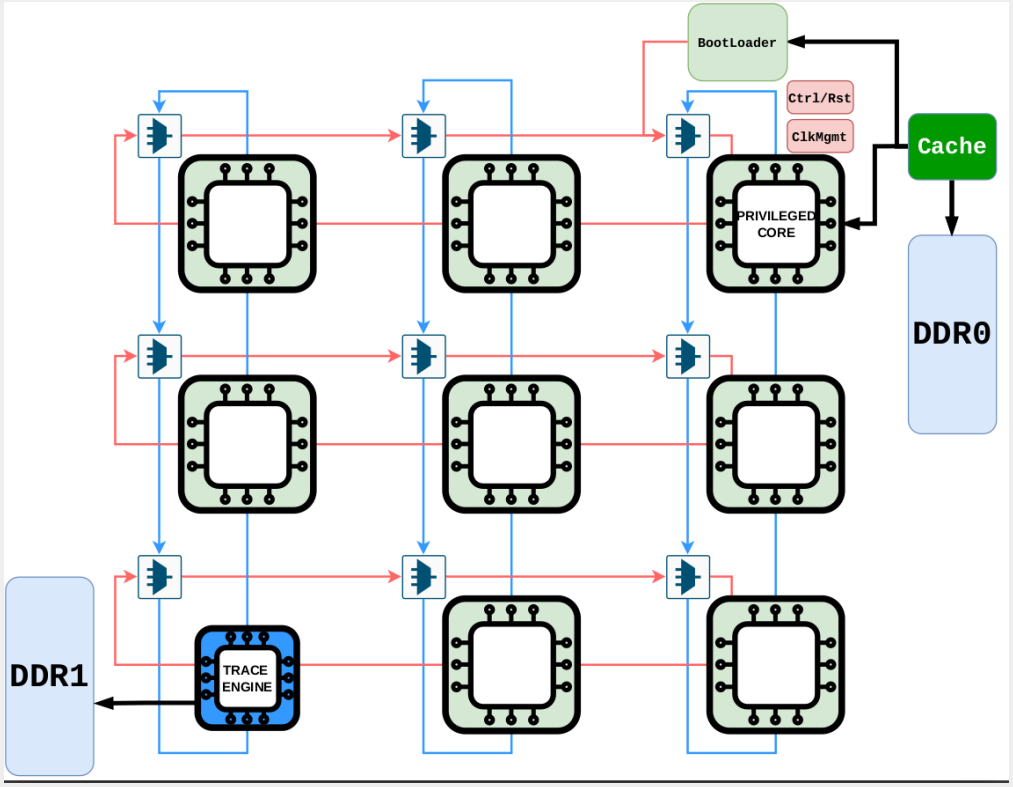
Instruction fusion opportunities in Manticore

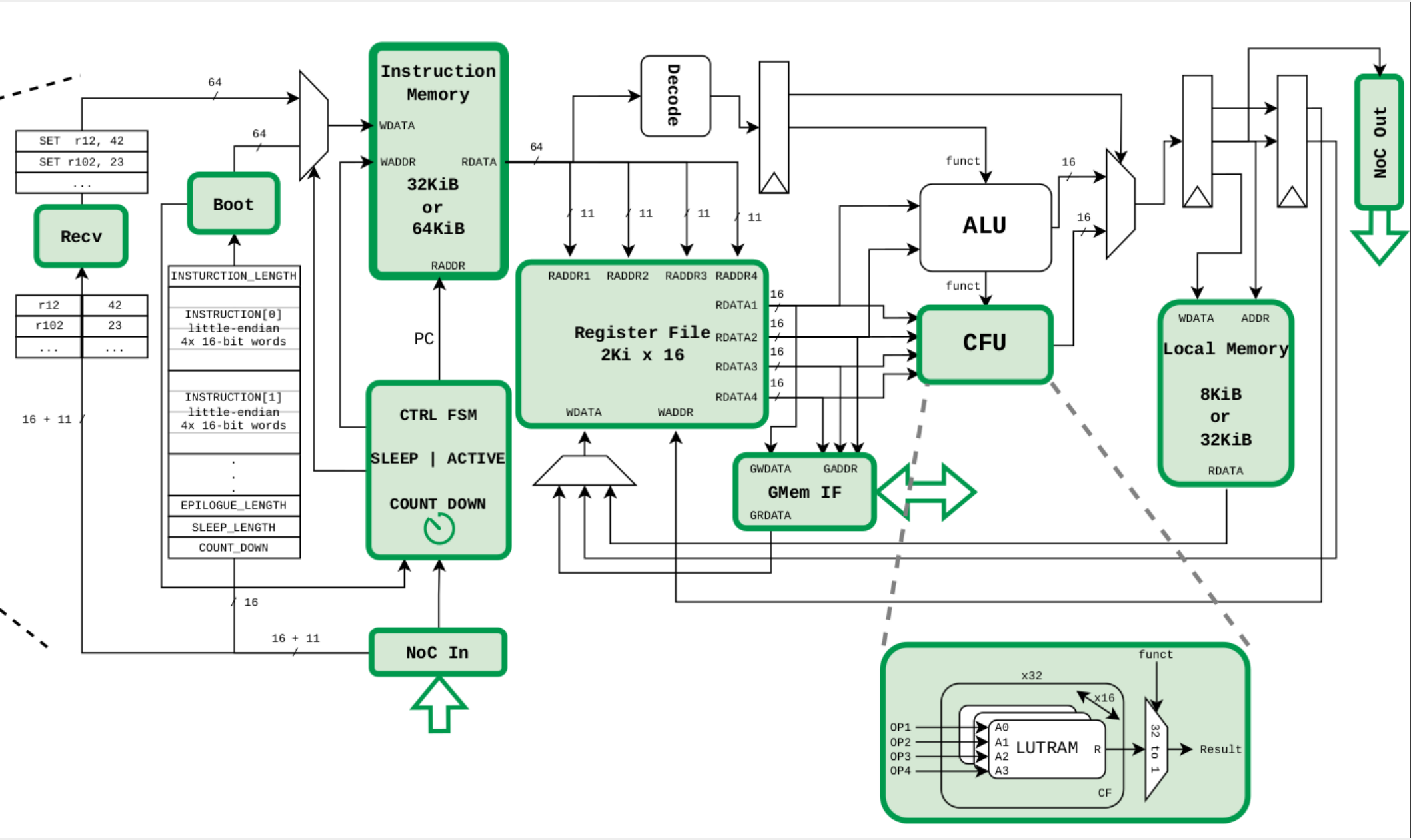
Bachelor semester project

Simon A. Marti, Spring 2022
Supervised by Sahand Kashani & Mahyar Emami

Manticore

- Goal : Accelerate RTL simulation with parallelization
- Run on FPGAs
- Array of custom CPUs
- Wimpy cores with a static schedule



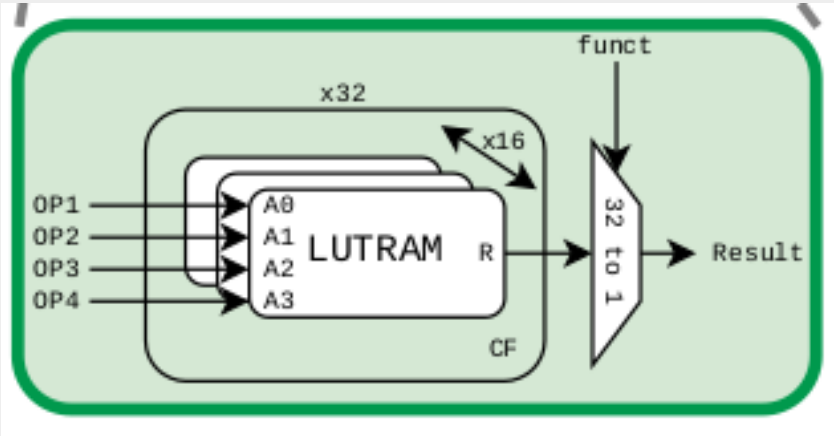

Instruction Memory
WDATA
WADDR
RDATA
32KiB or 64KiB
RADDR
Decode
Boot
Recv
SET r12, 42
SET r102, 23
...
r12 42
r102 23
INSTURCTION_LENGTH
INSTRUCTION[0] little-endian 4x 16-bit words
INSTRUCTION[1] little-endian 4x 16-bit words
EPILOGUE_LENGTH
SLEEP_LENGTH
COUNT_DOWN
PC
CTRL FSM
SLEEP | ACTIVE
COUNT DOWN
NoC In
16 + 11
Register File 2Ki x 16
RADDR1 RADDR2 RADDR3 RADDR4
RDATA1 RDATA2 RDATA3 RDATA4
WDATA WADDR
GMem IF
GWDATA GADDR
GRDATA
ALU
funct
CFU
Local Memory
8KiB or 32KiB
WDATA ADDR
RDATA
NoC Out
x32
x16
OP1 OP2 OP3 OP4
A0 A1 A2 A3
LUTRAM
R
CF
32 to 1
Result

Manticore compiler

- Verilog executable / Netlist
- Standard optimizations
  - Dead code elimination
  - Constant folding
  - Extract parallelism

Custom Instructions

- HW has support for up to 32 custom LUT
- 0 ≤ arity ≤ 4
- Replace logic instructions (AND, OR, XOR)
- Save simple instructions



Goal

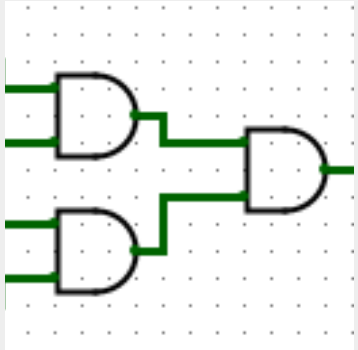
- Program with only normal instructions

  => Program with normal & custom instructions
- Find most profitable instructions to fuse
- Constraint : up to 32 different custom functions

4-inputs fusion examples

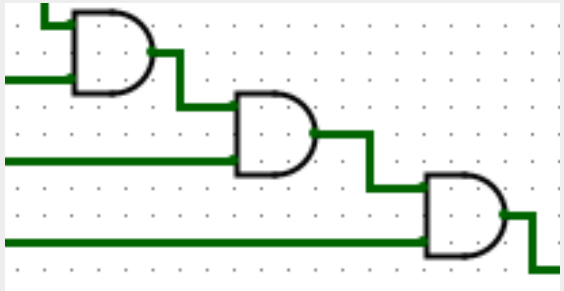
- “A-fusion”

```
AND w1, i1, i2;
AND w2, i3, i4;
AND w3, w1, w2;
```

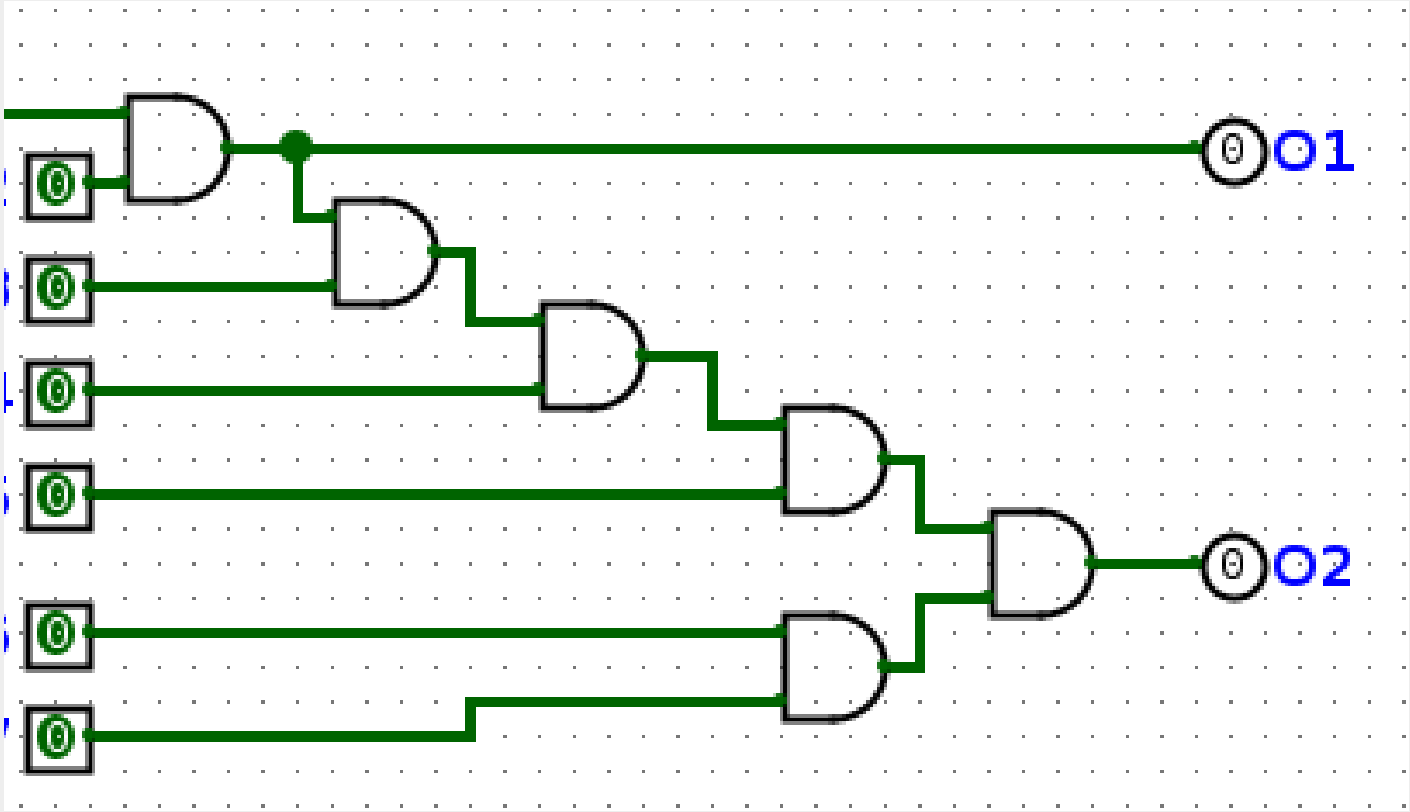


- “B-fusion”

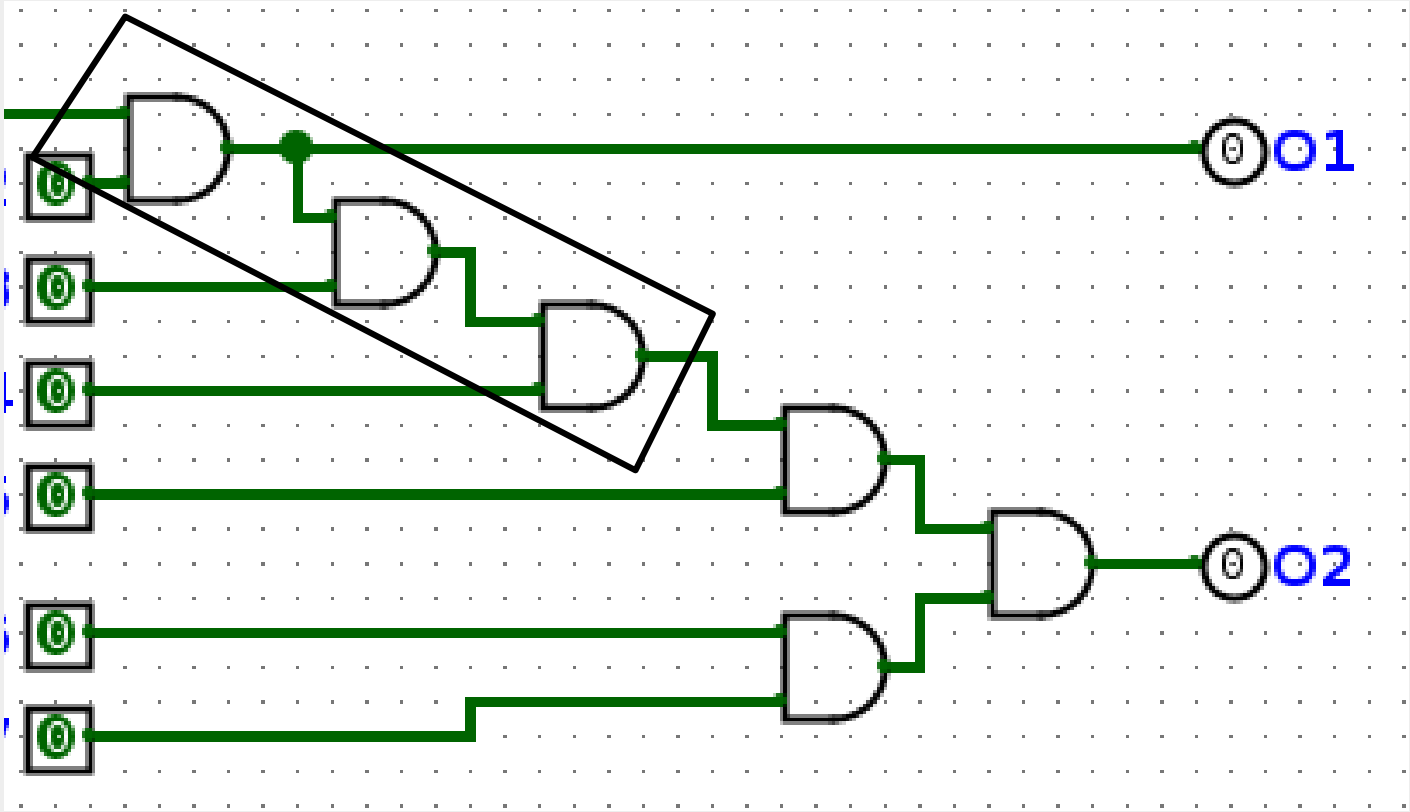
```
AND w1, i1, i2;
AND w2, w1, i3;
AND w3, w2, i4;
```



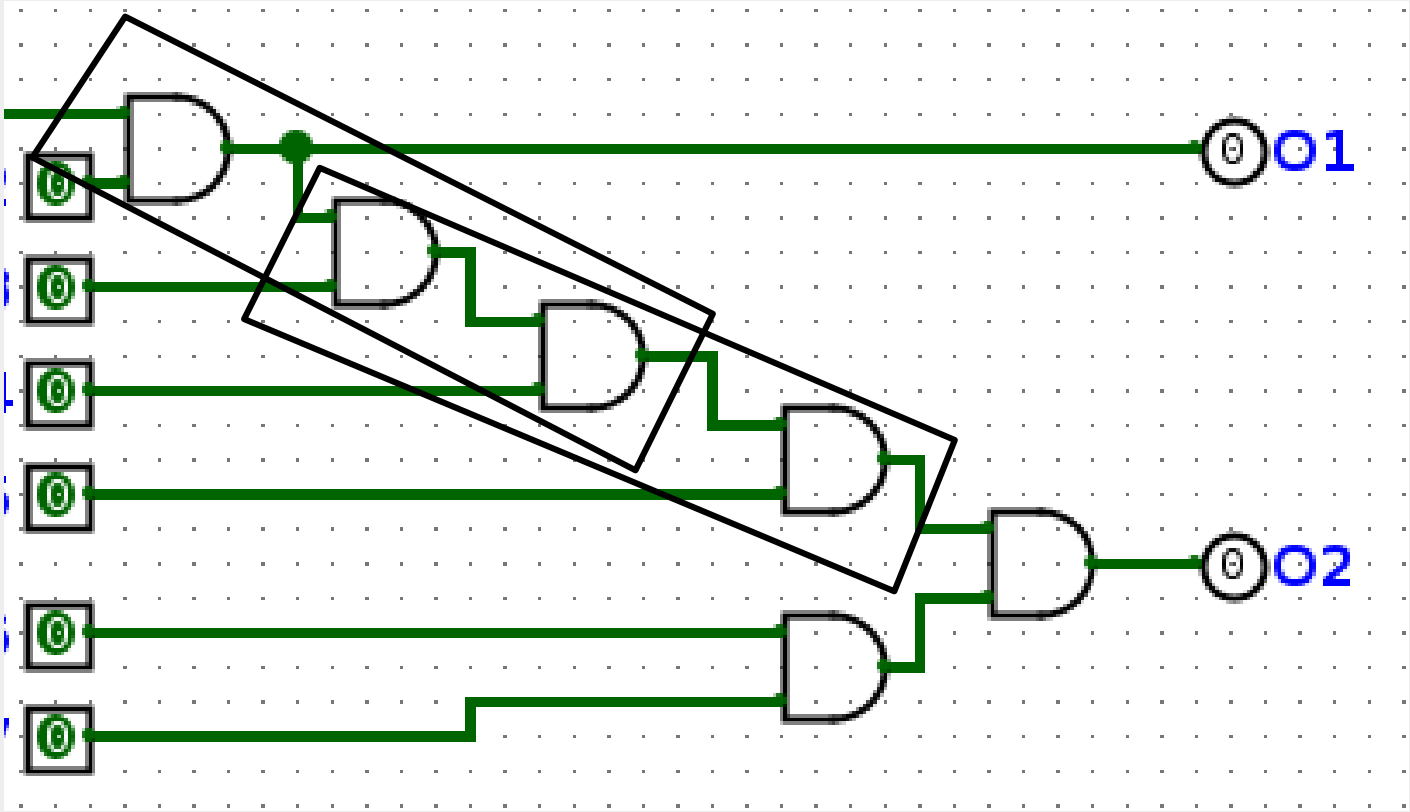
Fusion search



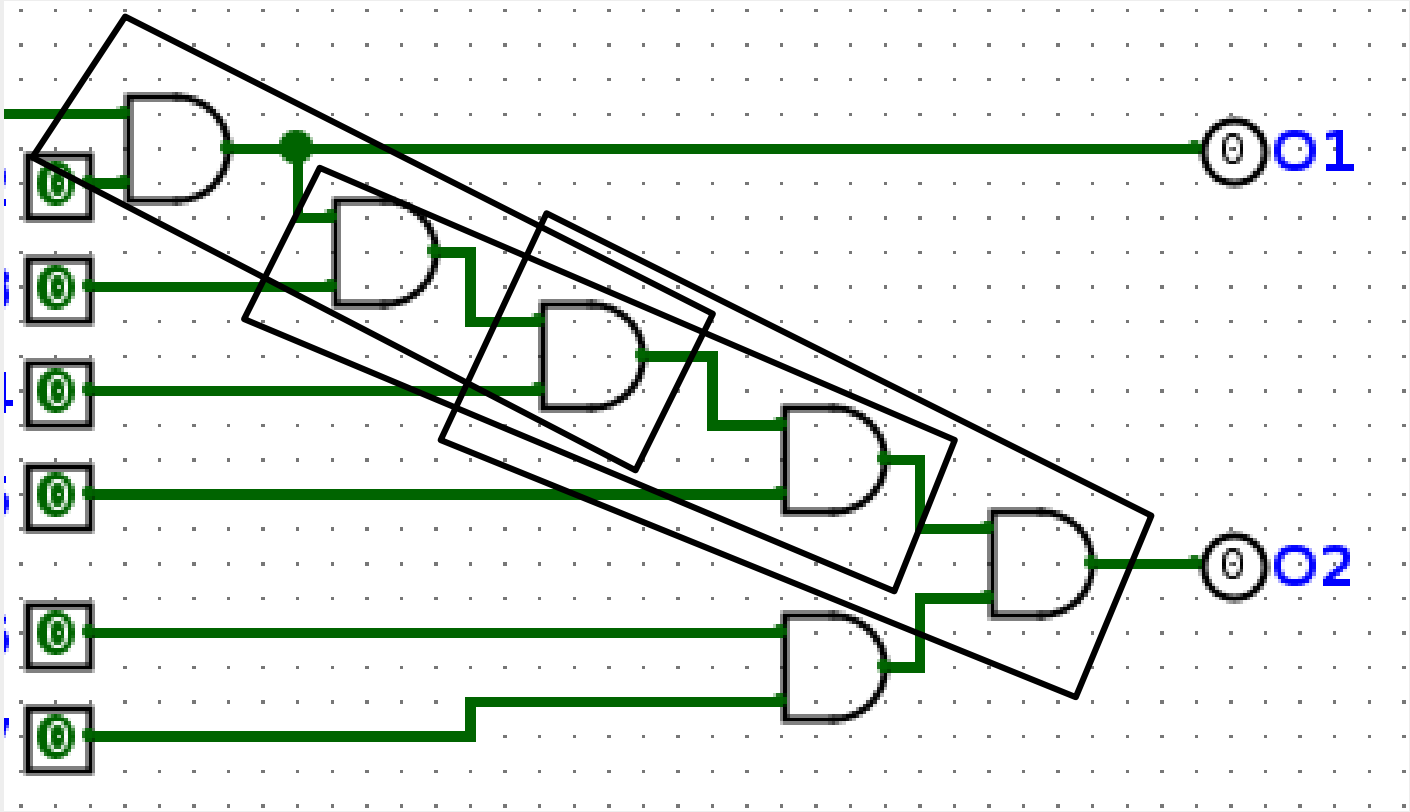
Fusion search



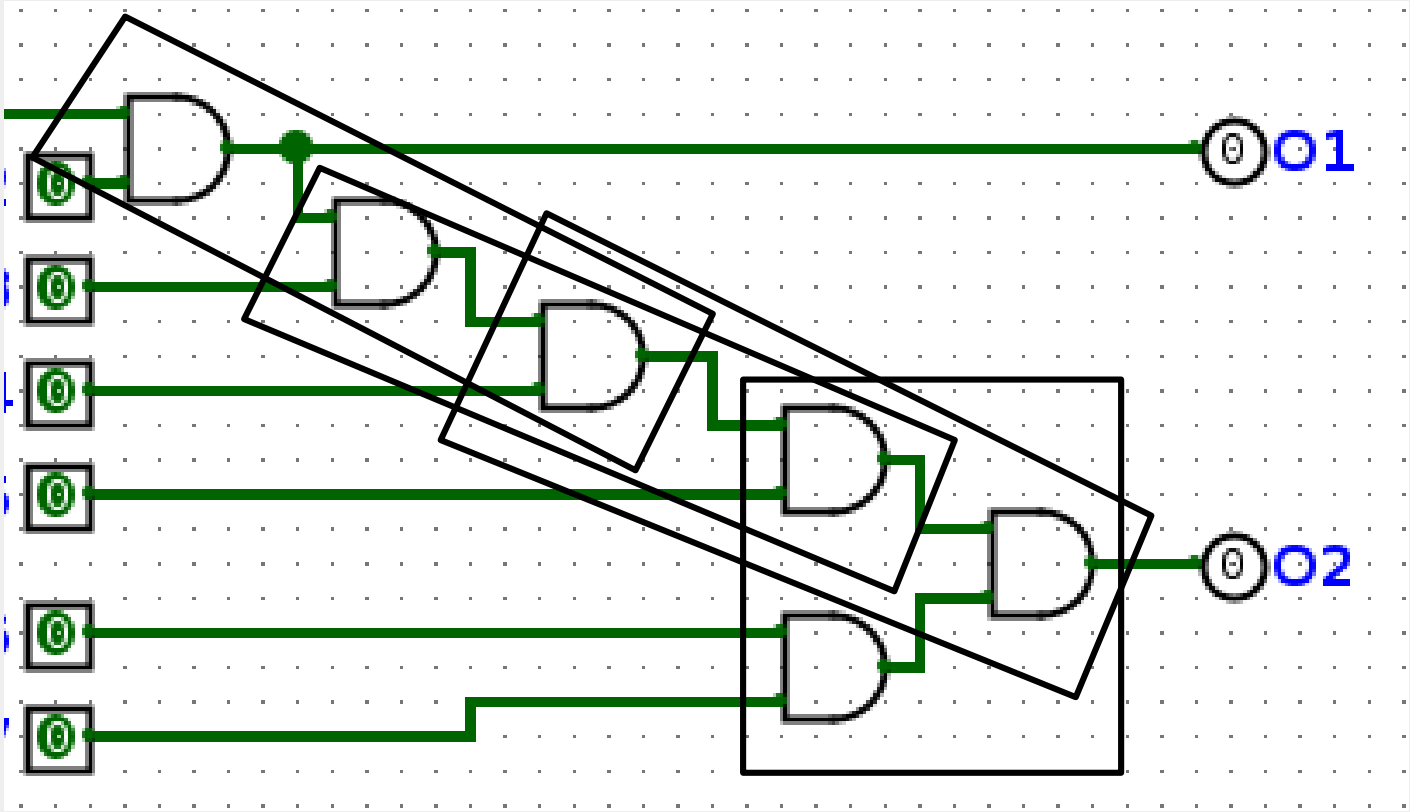
Fusion search



Fusion search

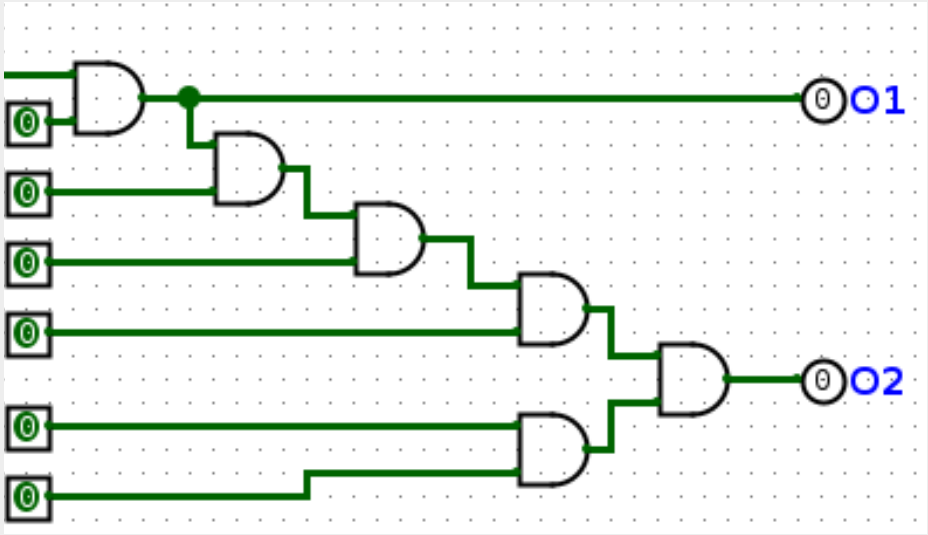


Fusion search



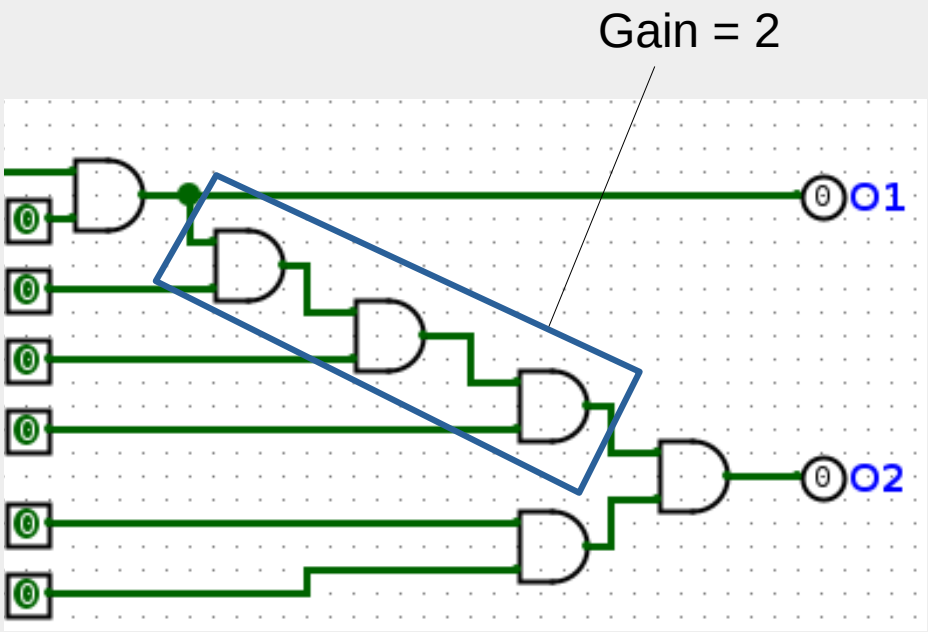
Fusion profitability

- Some fusion save more instructions than other



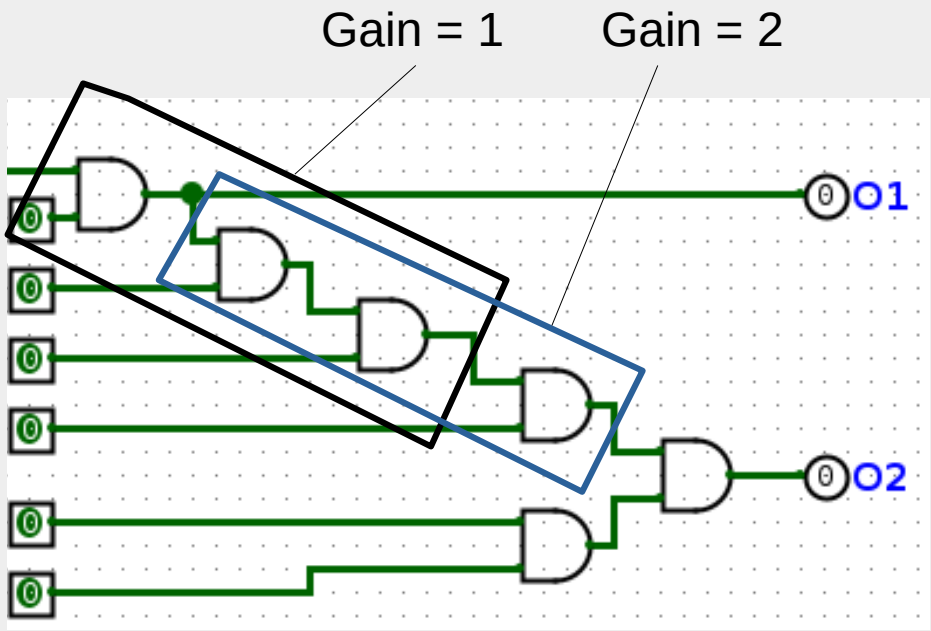
Fusion profitability

- Some fusion save more instructions than other



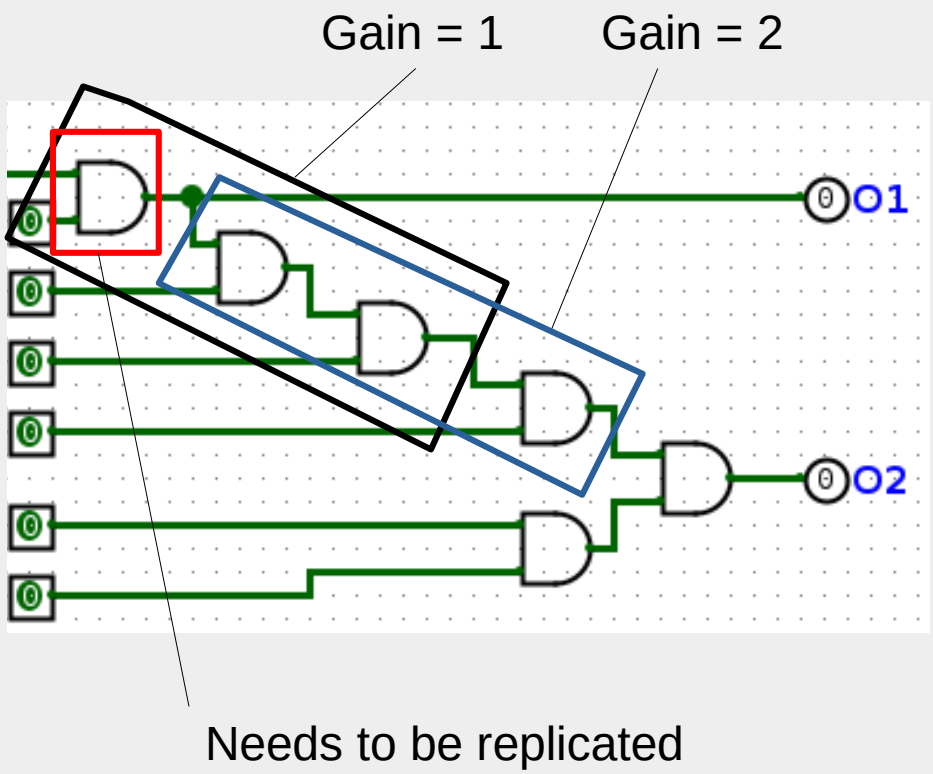
Fusion profitability

- Some fusion save more instructions than other

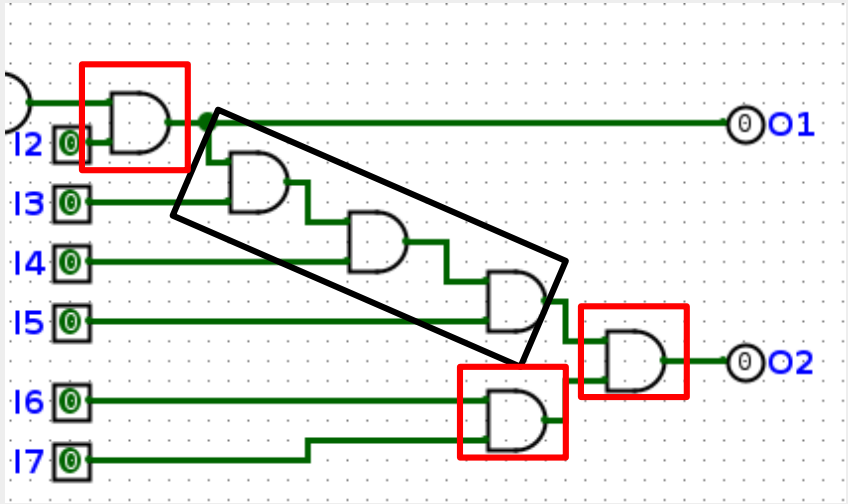


Fusion profitability

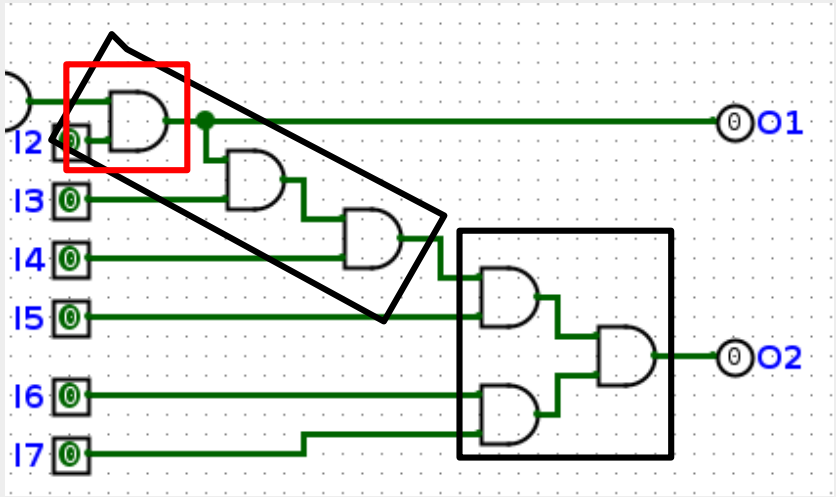
- Some fusion save more instructions than other



Fusion conflicts



Before : 6
After : 4
Gain = 2



Before : 6
After : 3
Gain = 3

Algorithm : fixed-point approach

```
0. build graph
do {
   1. find and select fusions to apply
   2. merge reused operands
   3. reduce arity
   4. merge arity-0 instructions
   5. collapse chains of arity-1 instructions
} 6. repeat until no more change in graph
```

0. Graph building

- Candidates for fusion : AND, OR, XOR
- Not considered : ADD, SLL, MUX, MOV, ...
- Three types of nodes :
  - Logic
  - Const
  - Name

```
AND w5, i8, i9;
AND w6, i10, i11;
AND w7, w5, w6;
MOV o3, w7;

AND w8, i12, i13;
SLL o5, w8, c1;

AND w9, i14, i15;
AND w10, w8, w9;
AND o4, w7, w10;
...
```

Pattern for fusion

w7 is reused

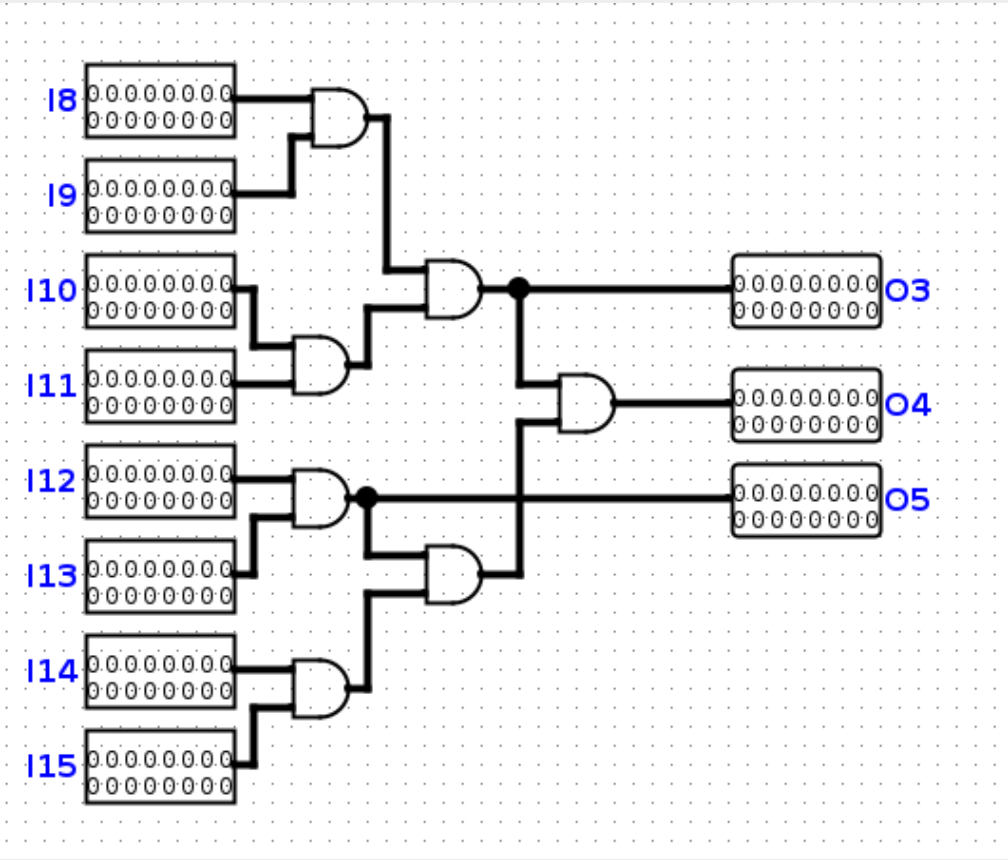
w8 is reused

```
AND w5, i8, i9;
AND w6, i10, i11;
AND w7, w5, w6;
MOV o3, w7;

AND w8, i12, i13;
SLL o5, w8, c1;

AND w9, i14, i15;
AND w10, w8, w9;
AND o4, w7, w10;
...
```

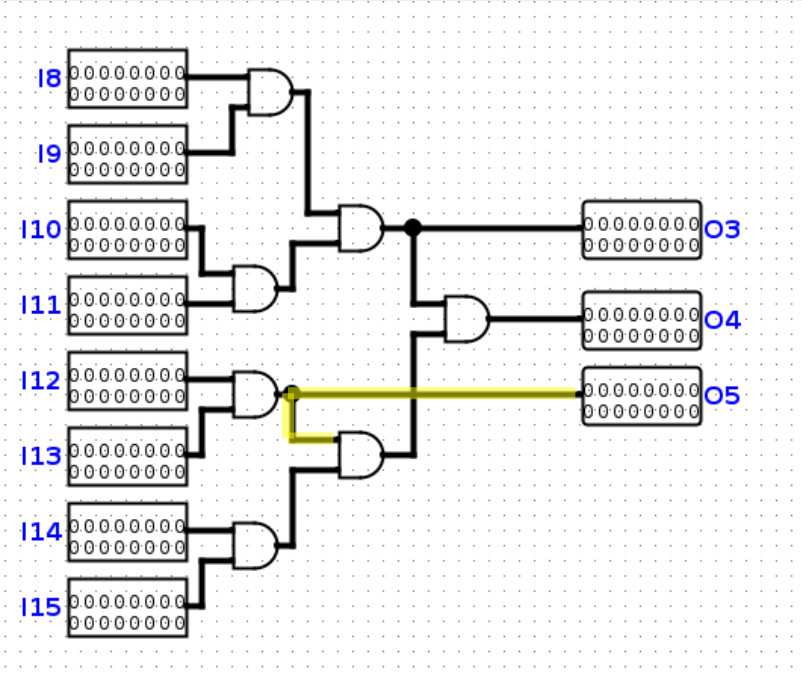
Pattern for fusion

w7 is reused

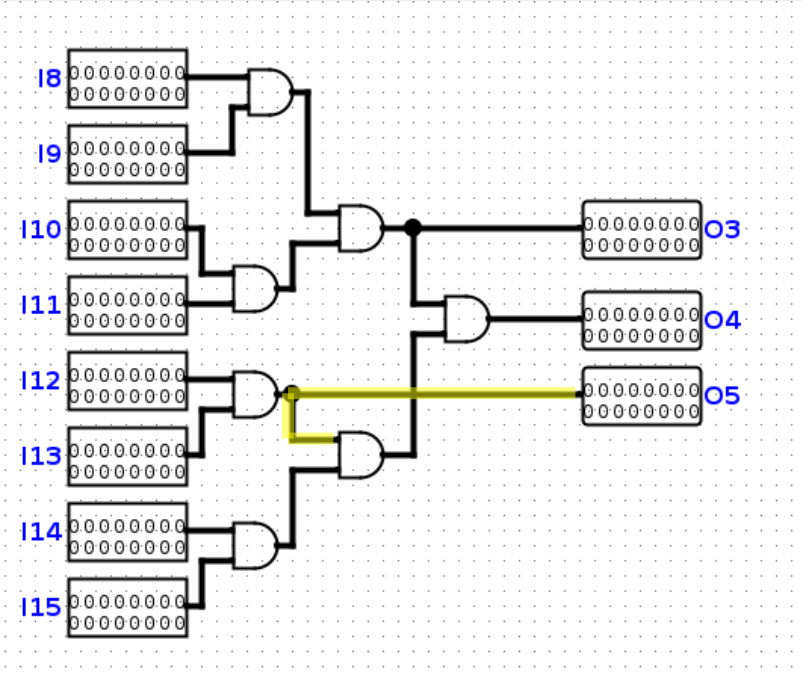
w8 is reused

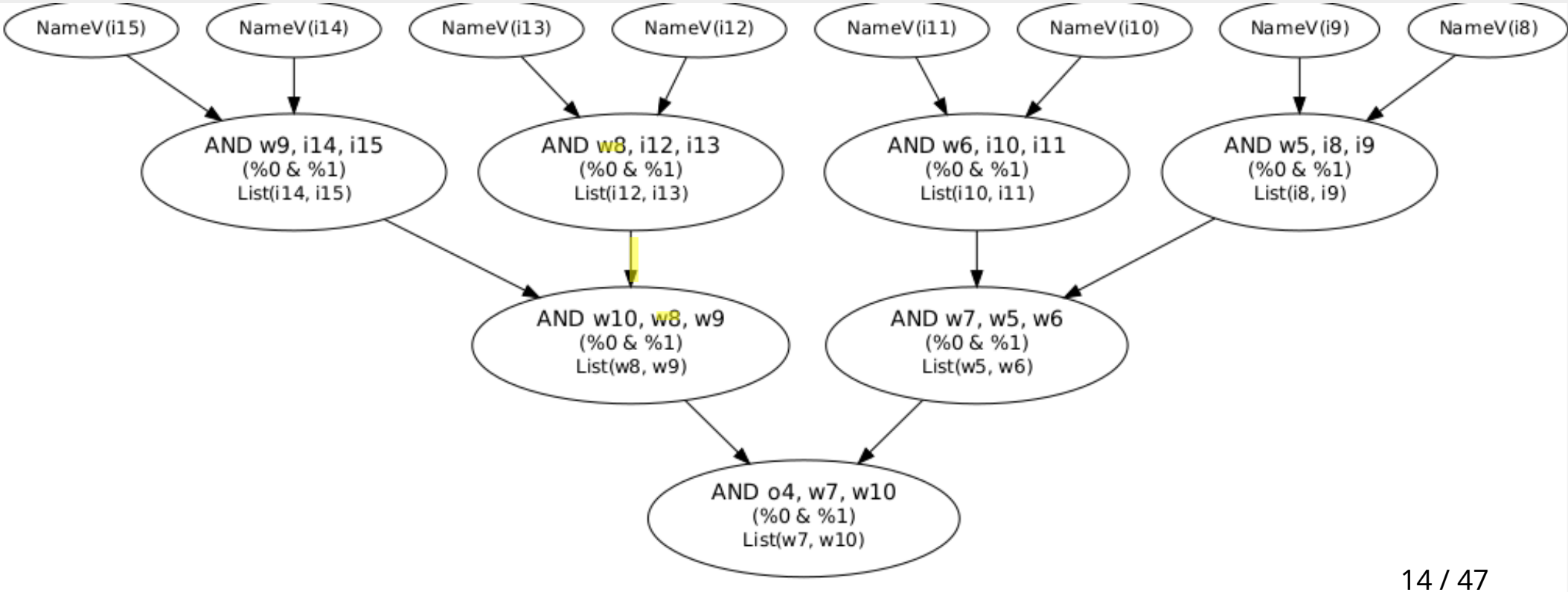


```
AND w5, i8, i9;
AND w6, i10, i11;
AND w7, w5, w6;
MOV o3, w7;

AND w8, i12, i13;
SLL o5, w8, c1;

AND w9, i14, i15;
AND w10, w8, w9;
AND o4, w7, w10;
...
```



```
AND w5, i8, i9;
AND w6, i10, i11;
AND w7, w5, w6;
MOV o3, w7;

AND w8, i12, i13;
SLL o5, w8, c1;

AND w9, i14, i15;
AND w10, w8, w9;
AND o4, w7, w10;
...
```





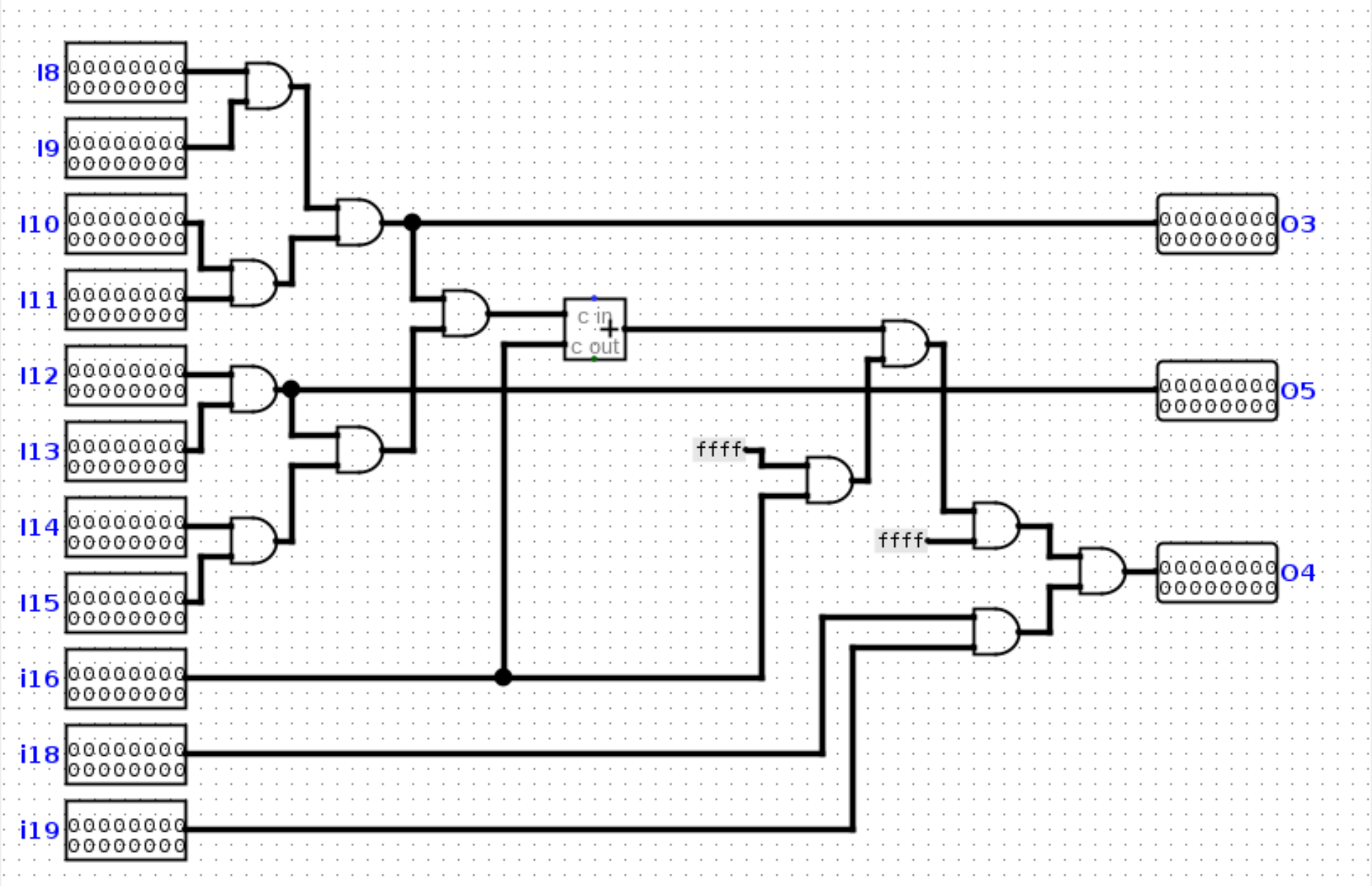
1.1 Fusion pattern-based enumeration

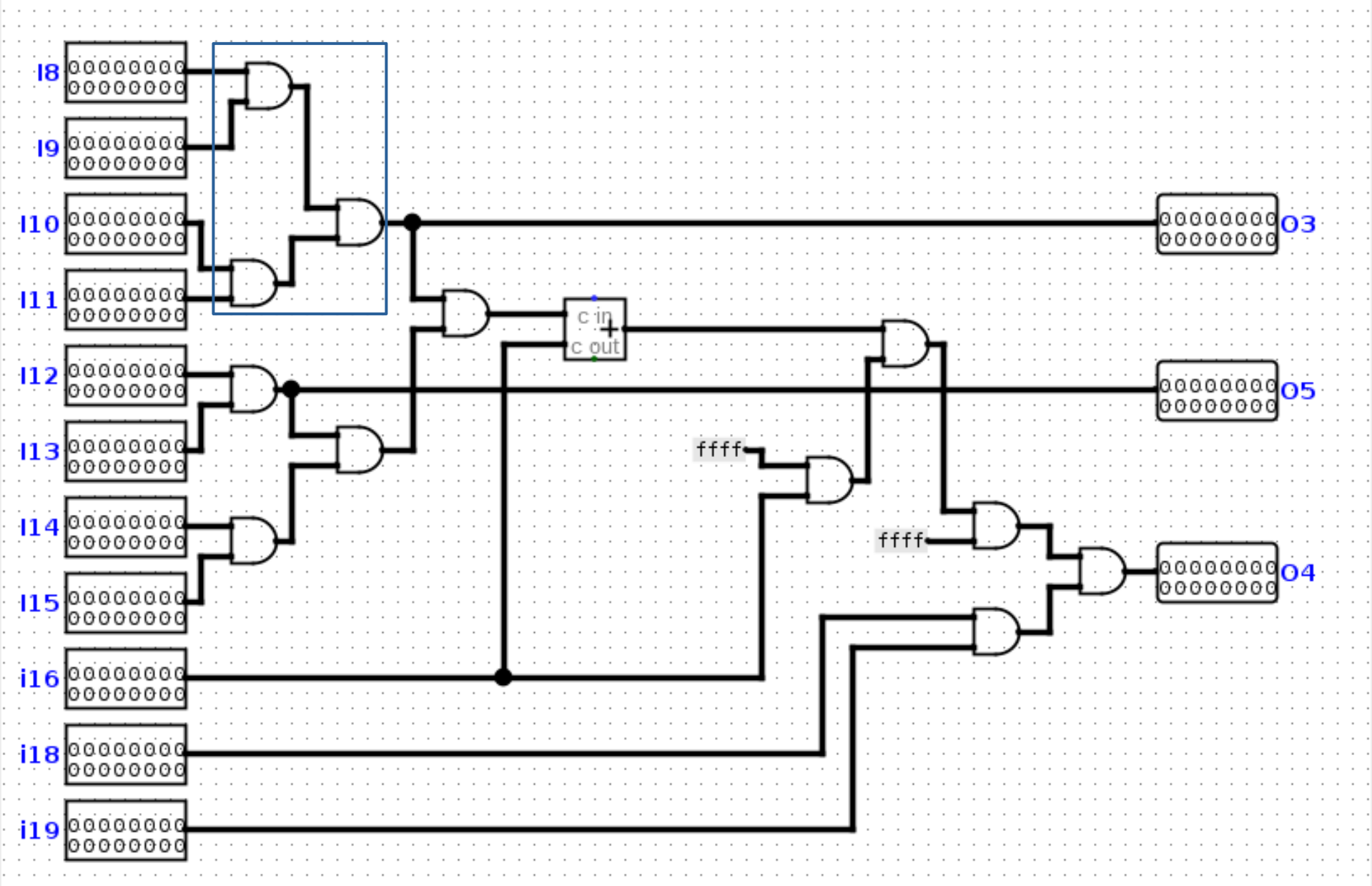
- Pattern matching approach

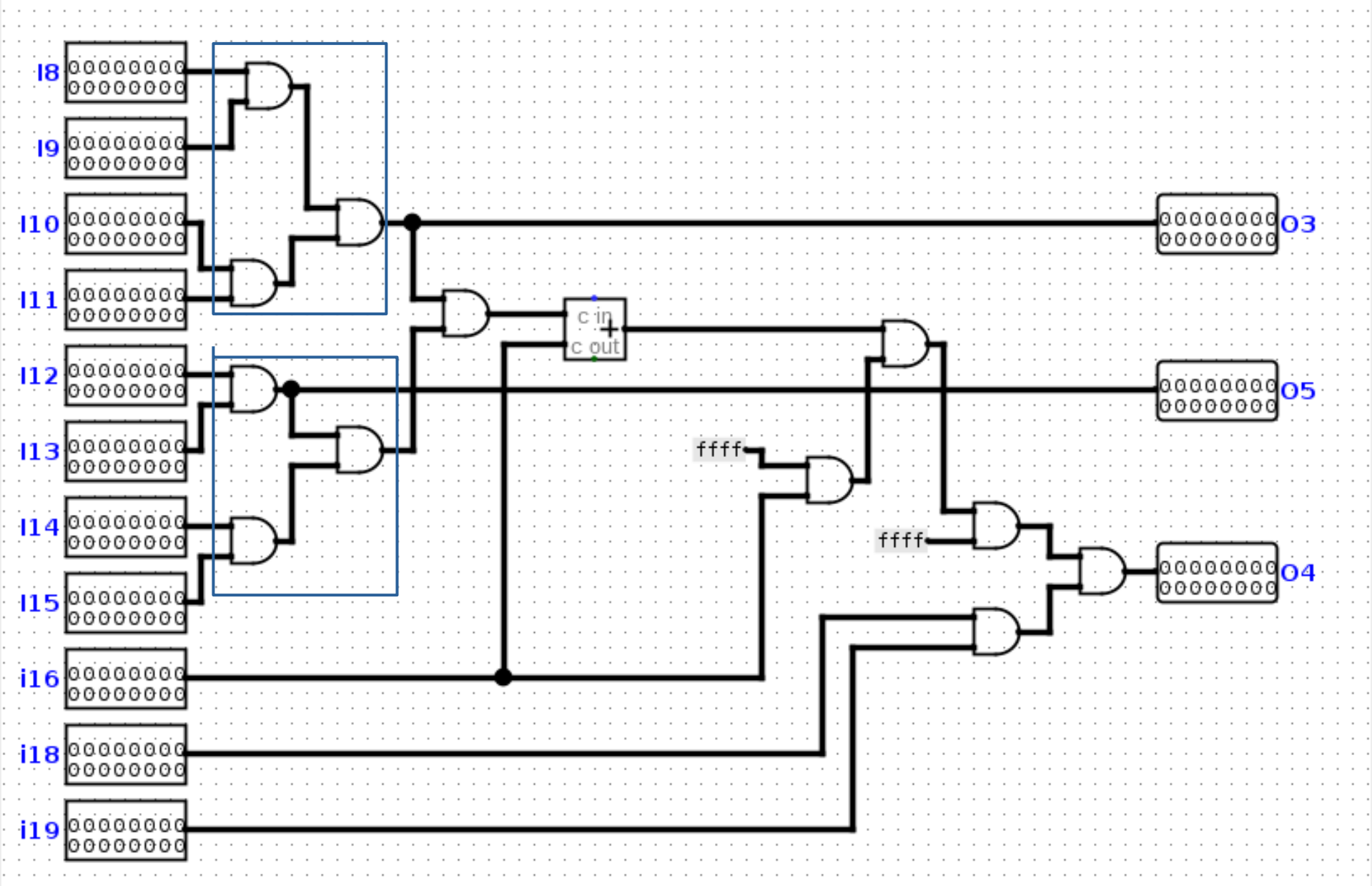
```
for i in body.arity2Instructions :
  for j in i.inputs :
    for k in j.inputs :
      if (i, j, k) forms a fusion pattern :
        record (i, j, k)
```

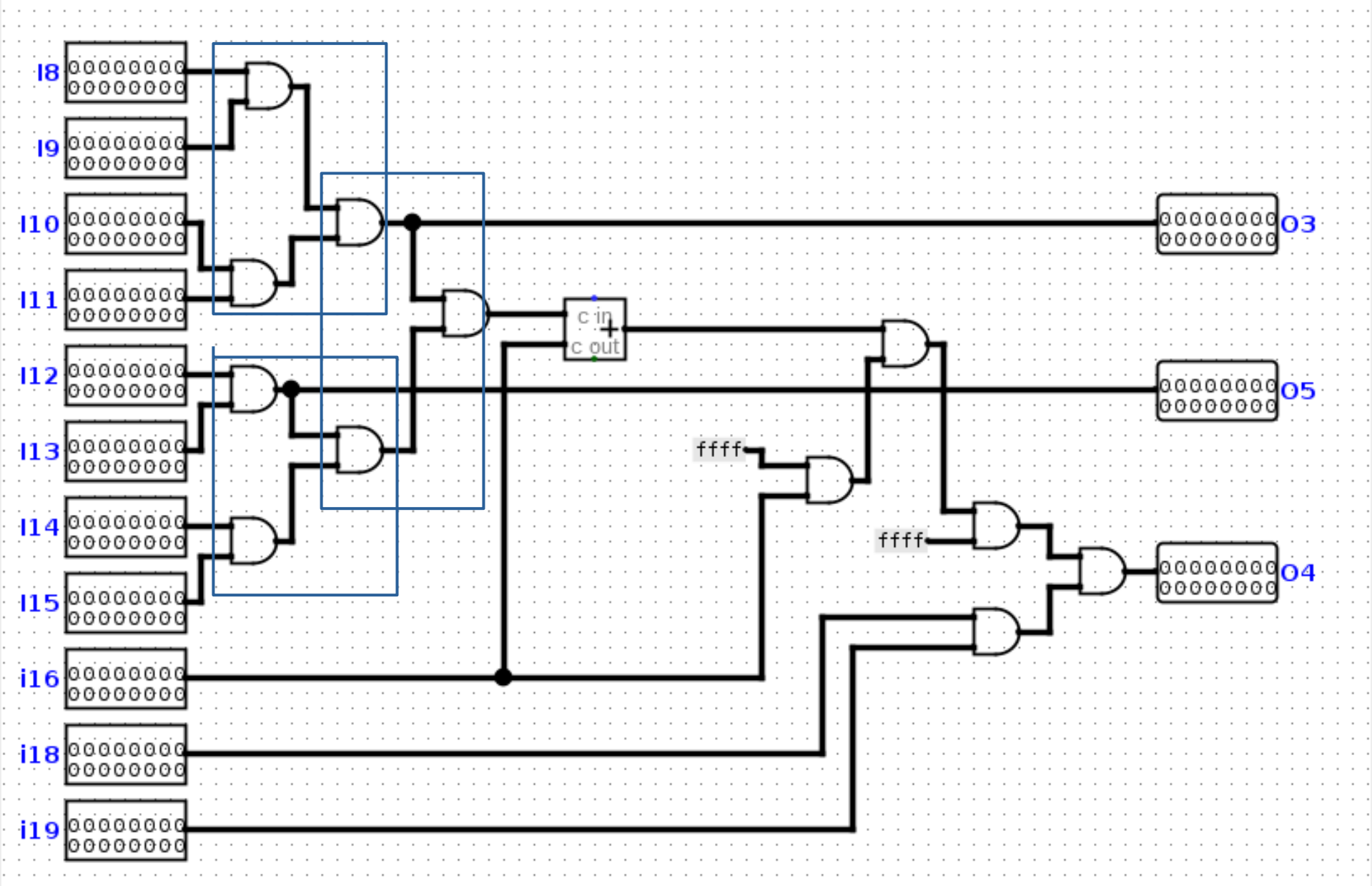
1.2 Collisions and collision zones

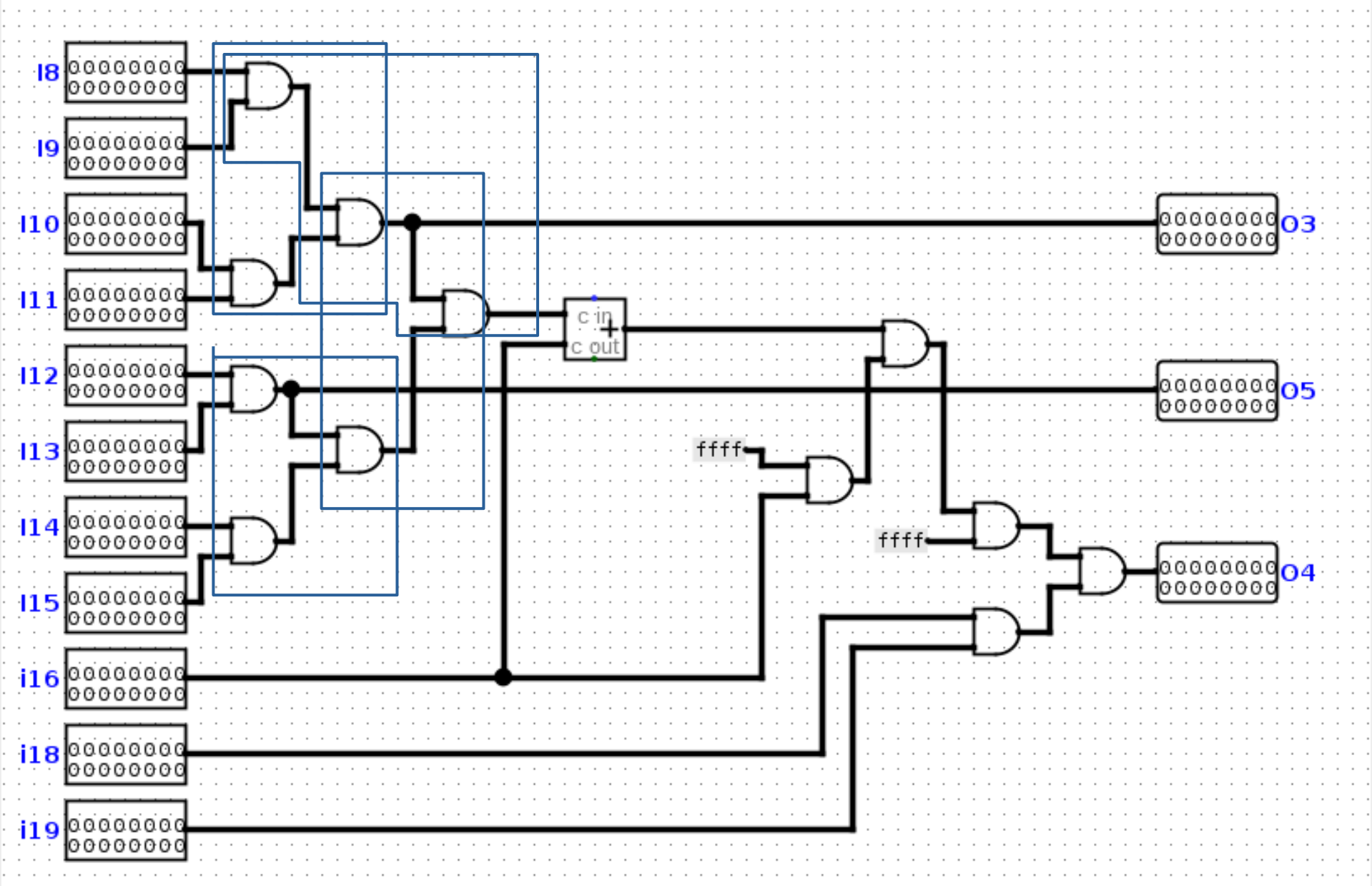
- Each fusion exclude some others
- “Collision zone” = isolated portion of the program where some fusions overlap
- Typical size : 1 - 300

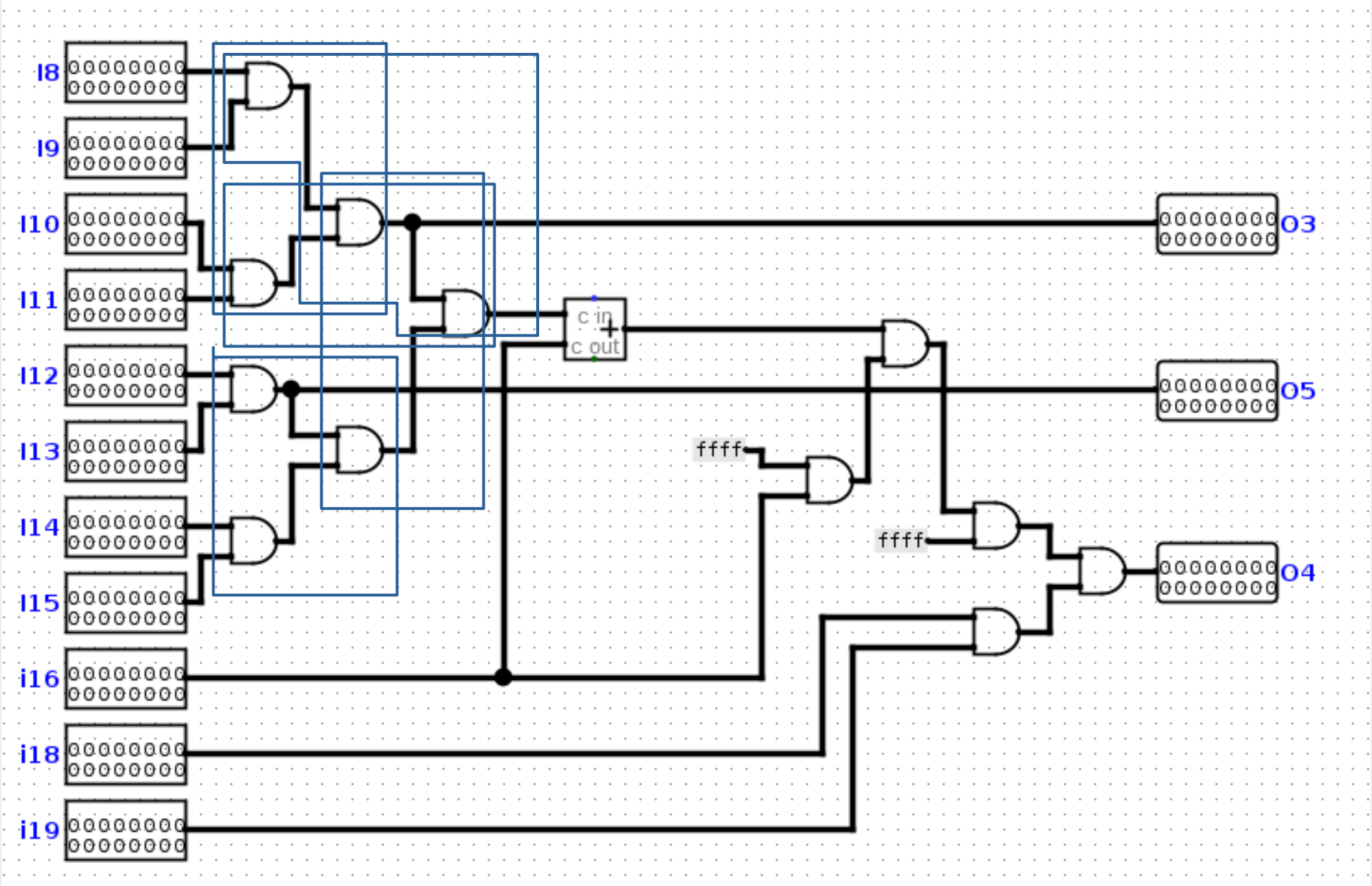

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

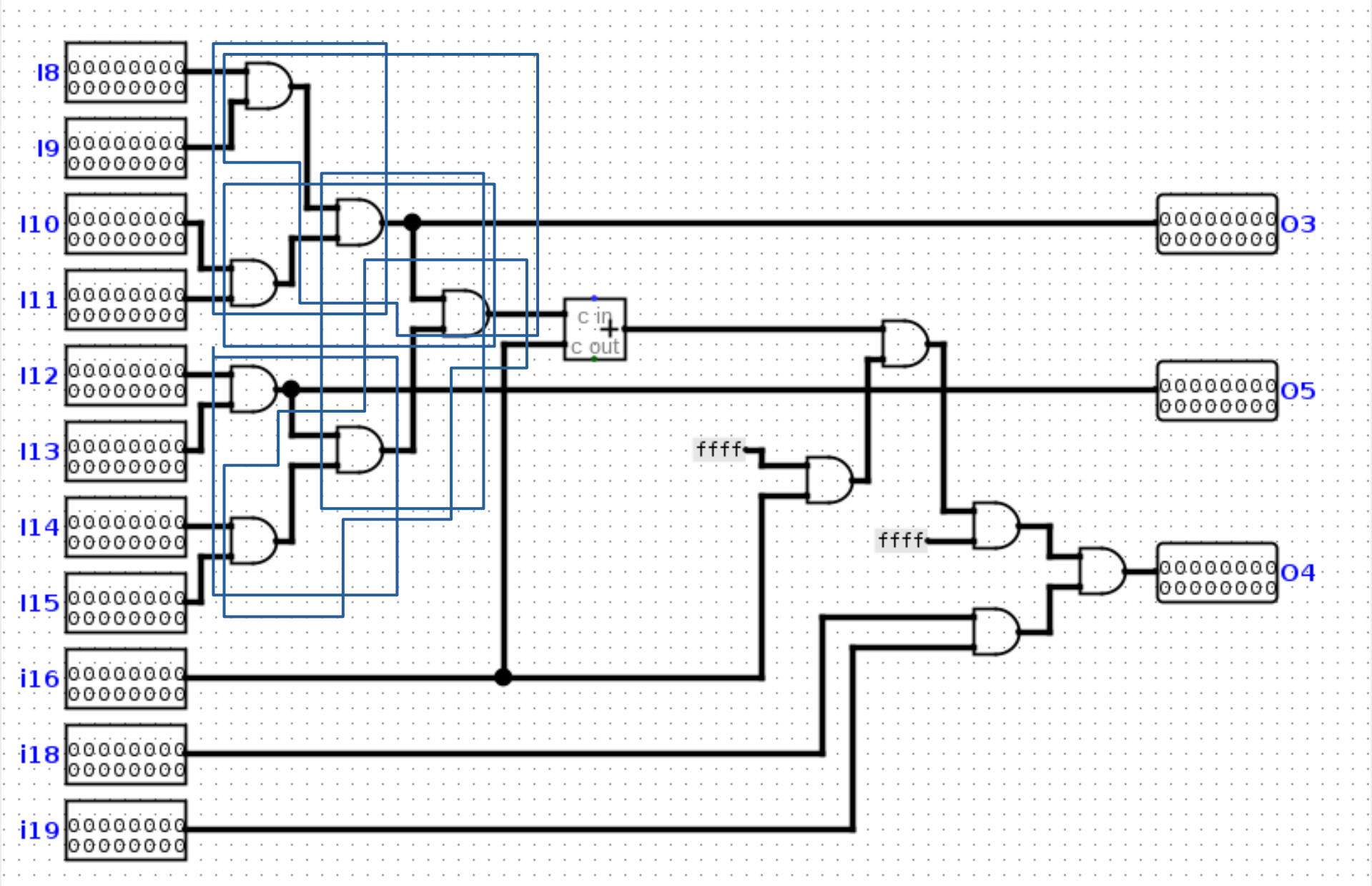

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

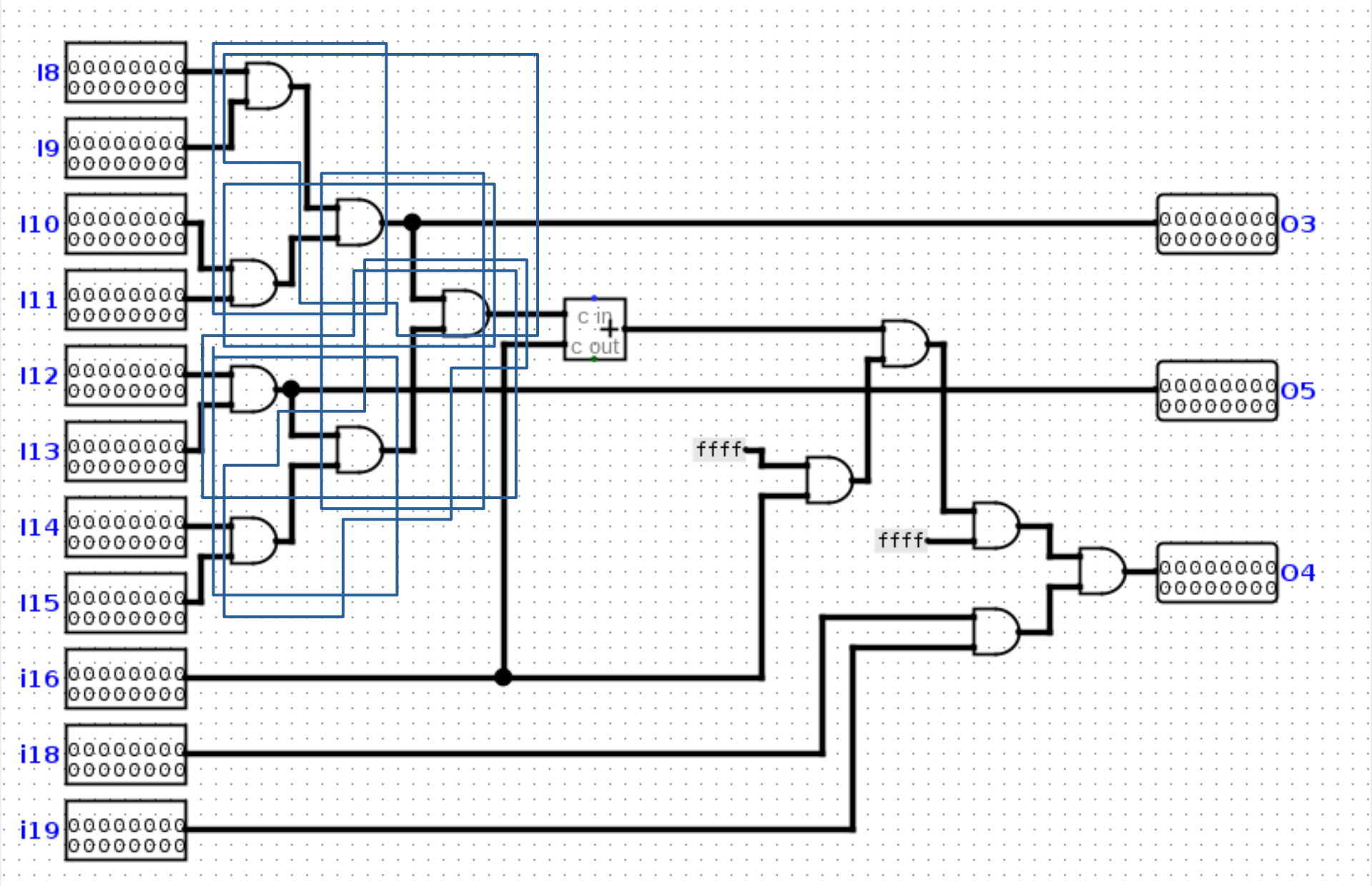

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

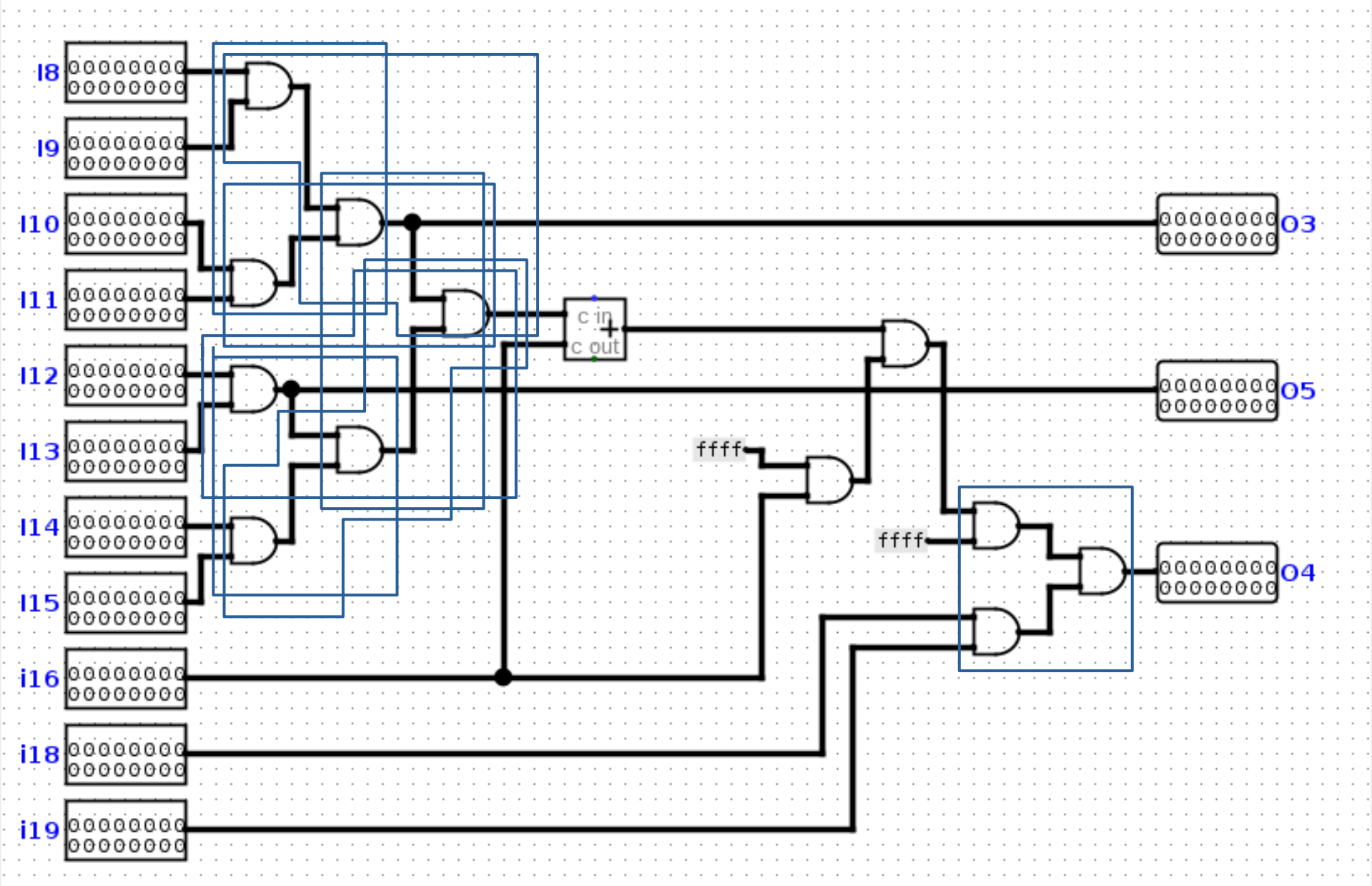

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

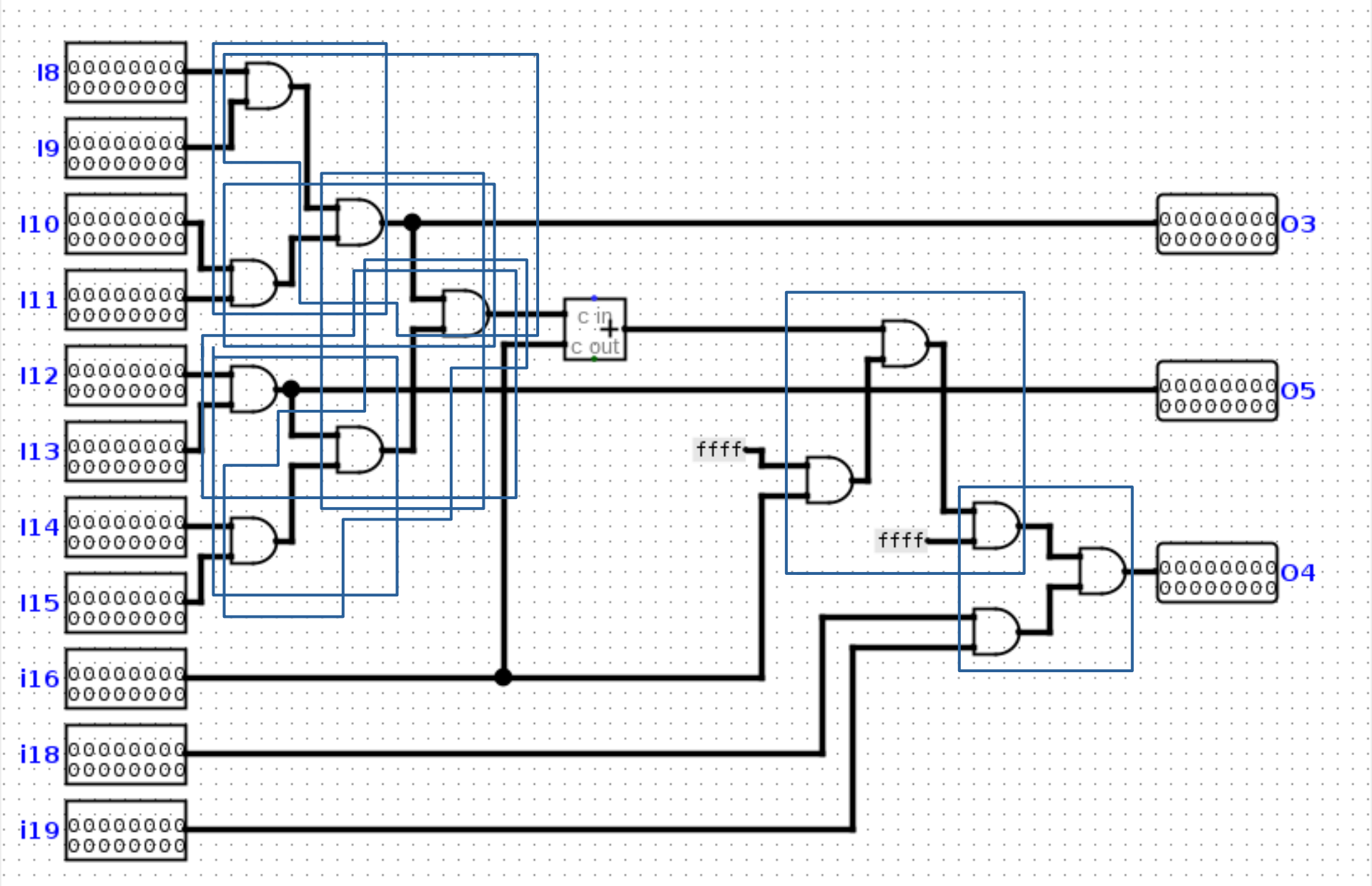

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

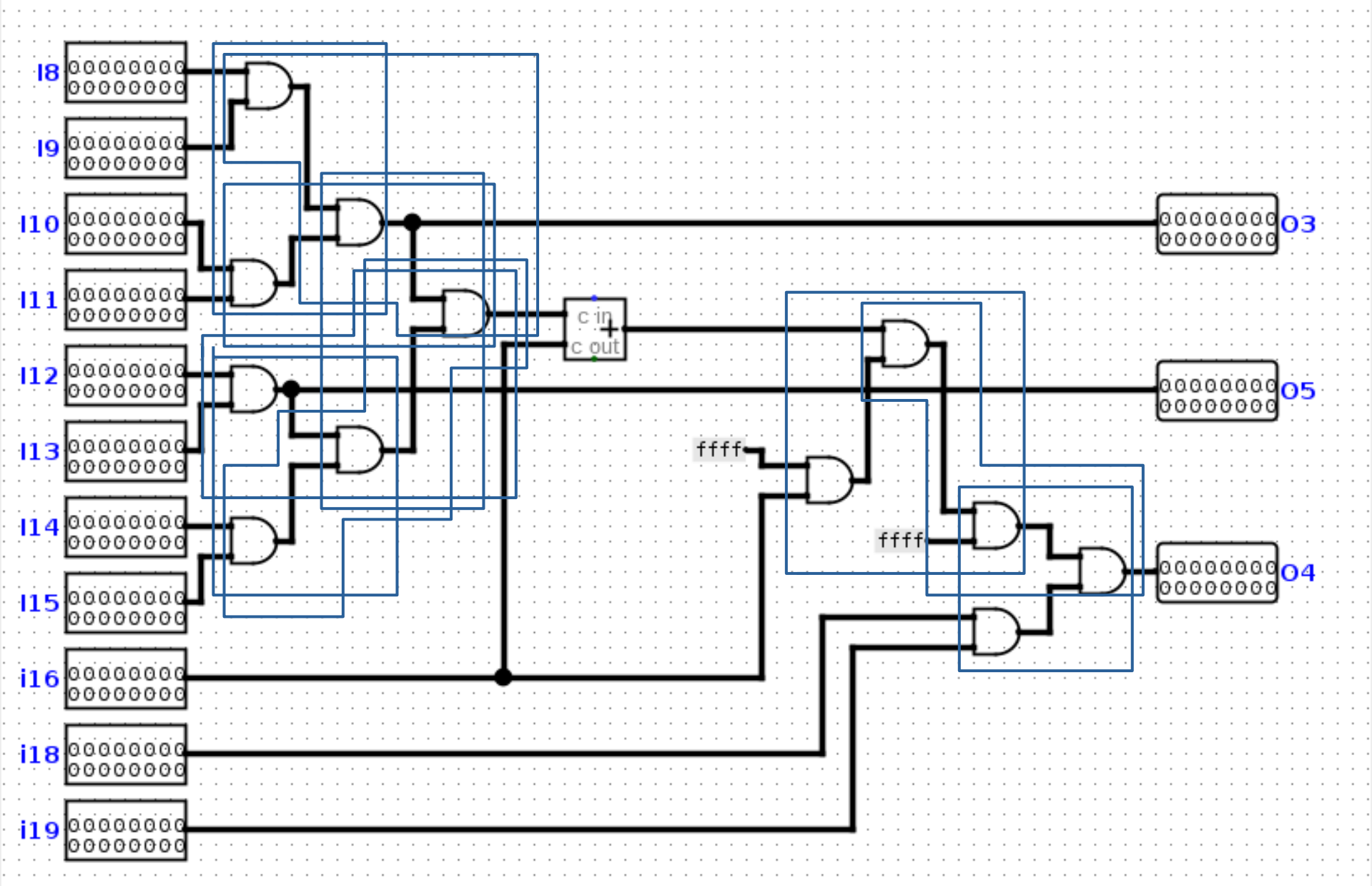

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4


I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

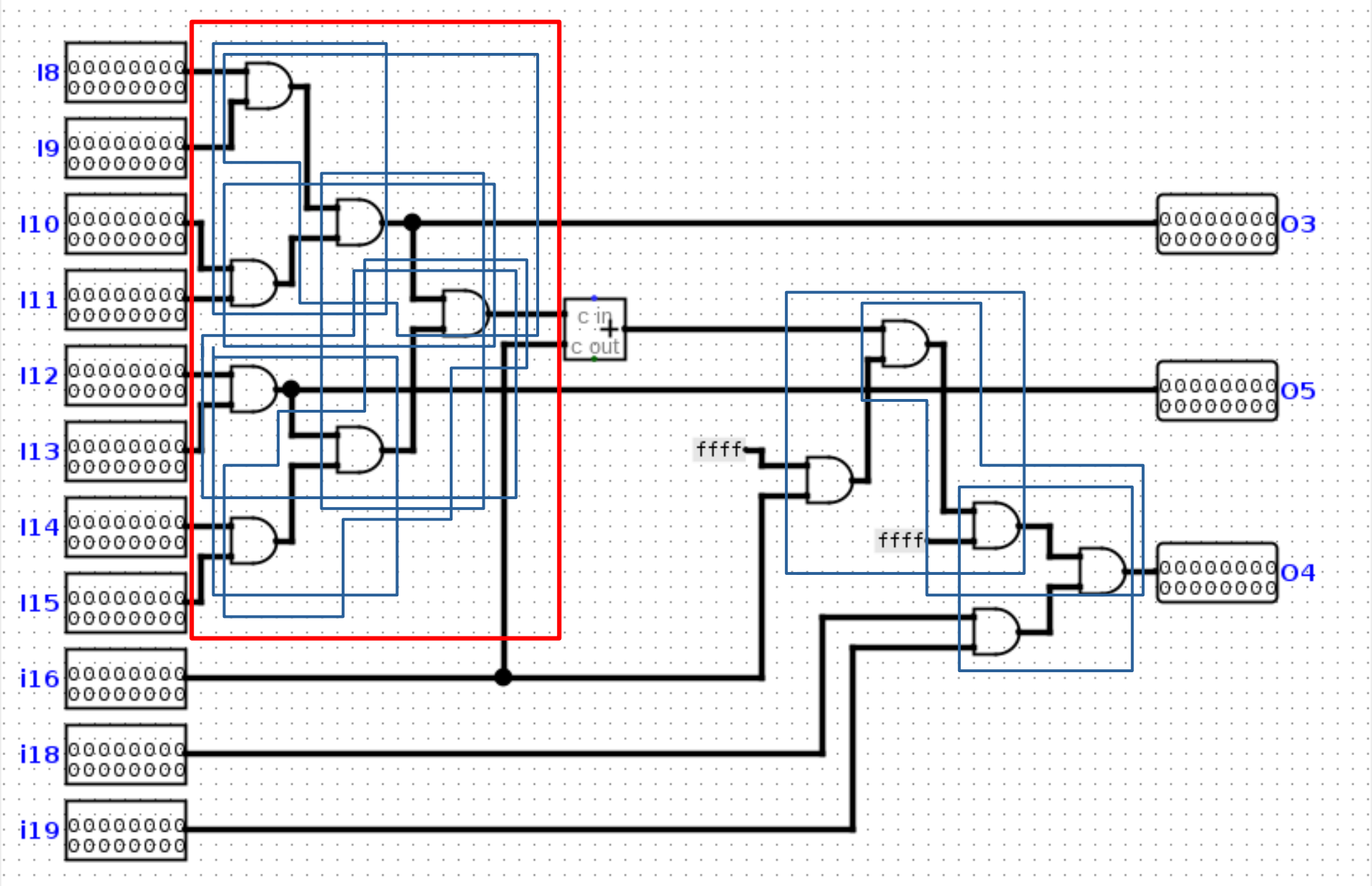

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

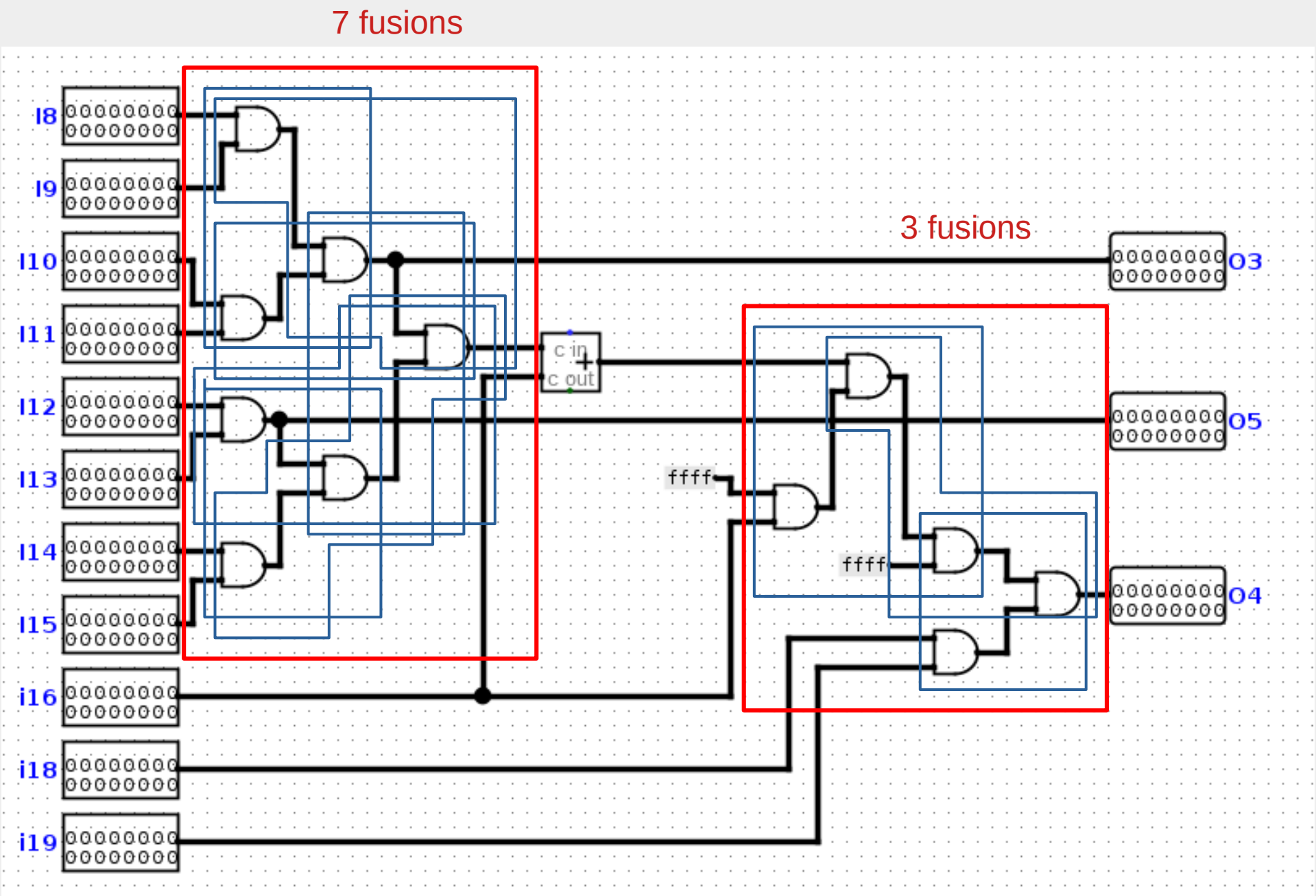

I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4


I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4


I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

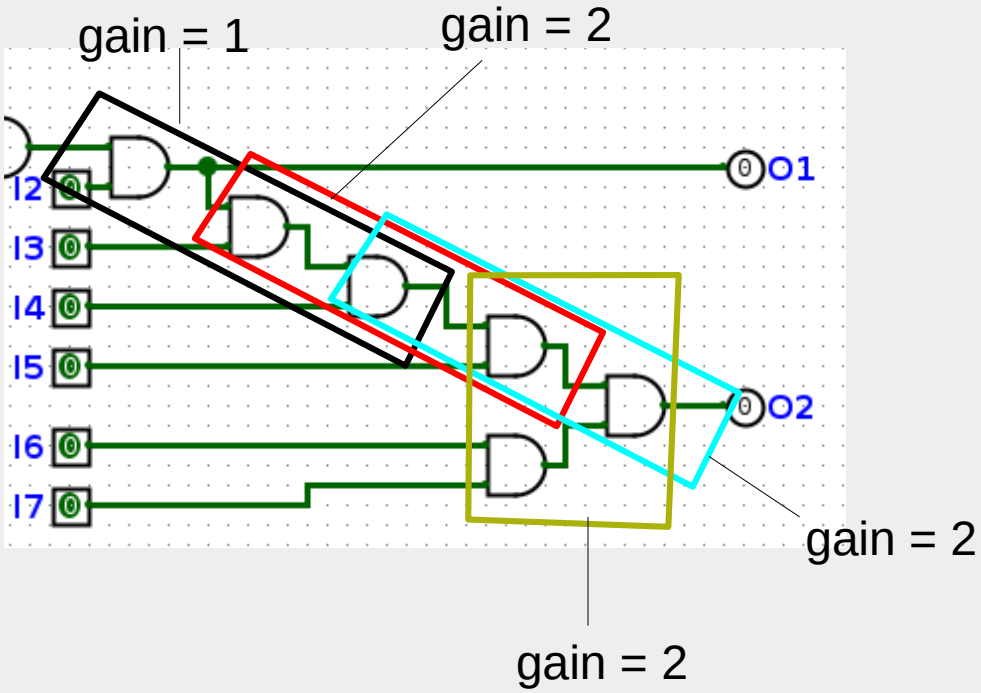
7 fusions


I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4


7 fusions
3 fusions
I8
I9
I10
I11
I12
I13
I14
I15
i16
i18
i19
c in
c out
ffff
ffff
O3
O5
O4

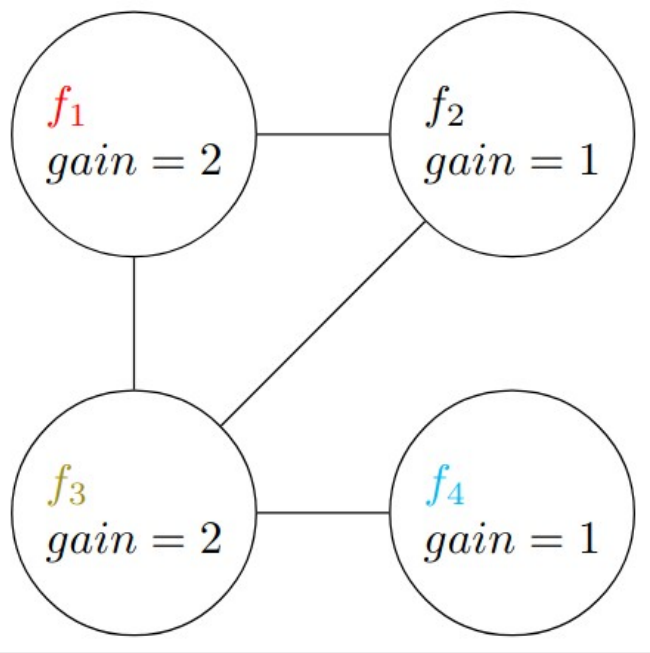
1.3 Fusion selection

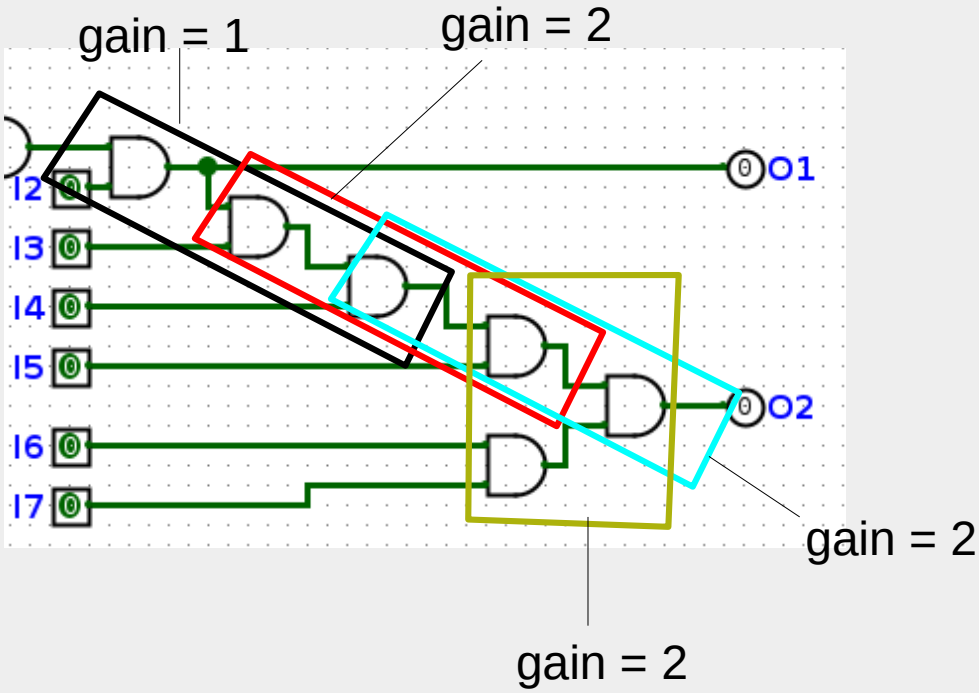
```
AND w0, w22, i2;
AND w1, w0, i3;
AND w2, w1, i4;
AND w3, w2, i5;
AND w4, i6, i7;
AND o2, w3, w4;
```



1.3 Fusion selection

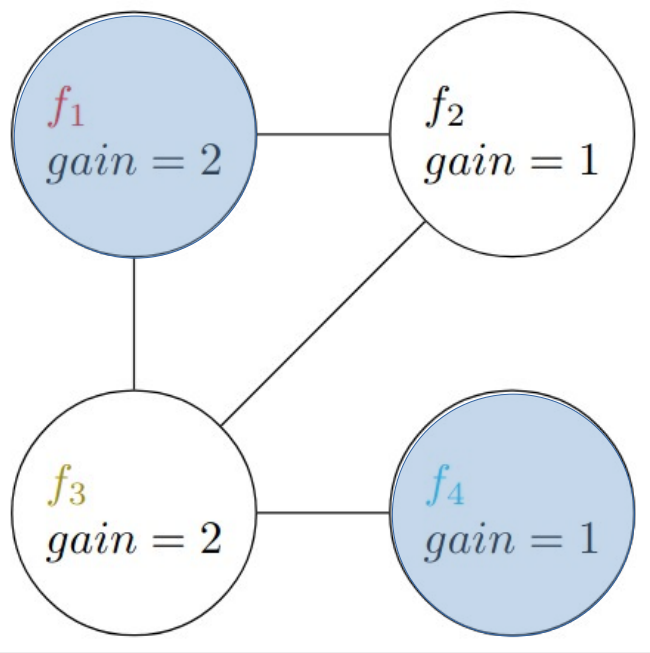
```
AND w0, w22, i2;
AND w1, w0, i3;
AND w2, w1, i4;
AND w3, w2, i5;
AND w4, i6, i7;
AND o2, w3, w4;
```

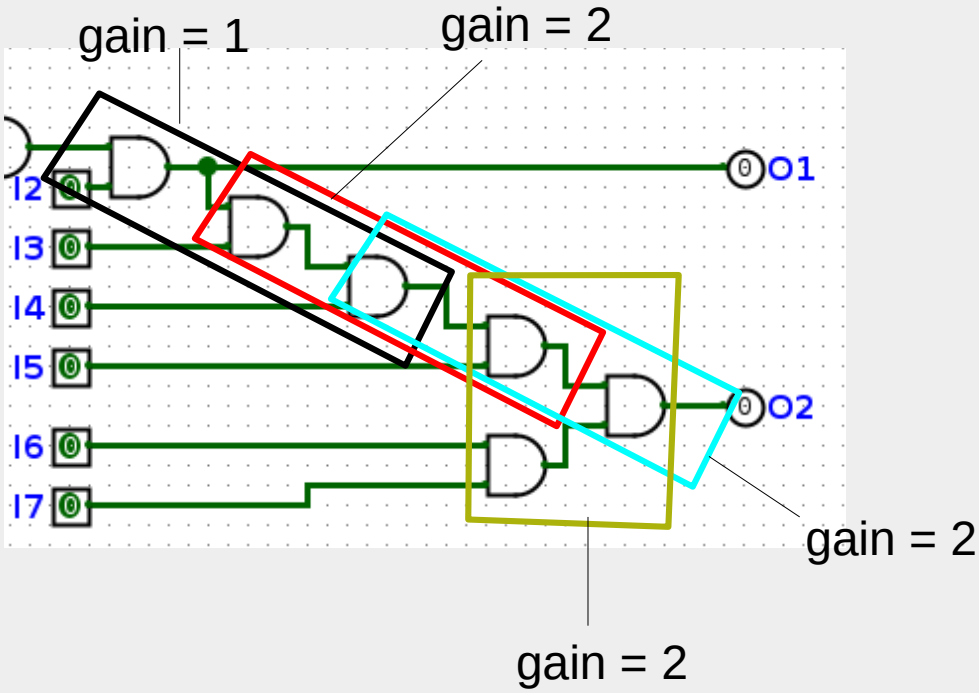




1.3 Fusion selection

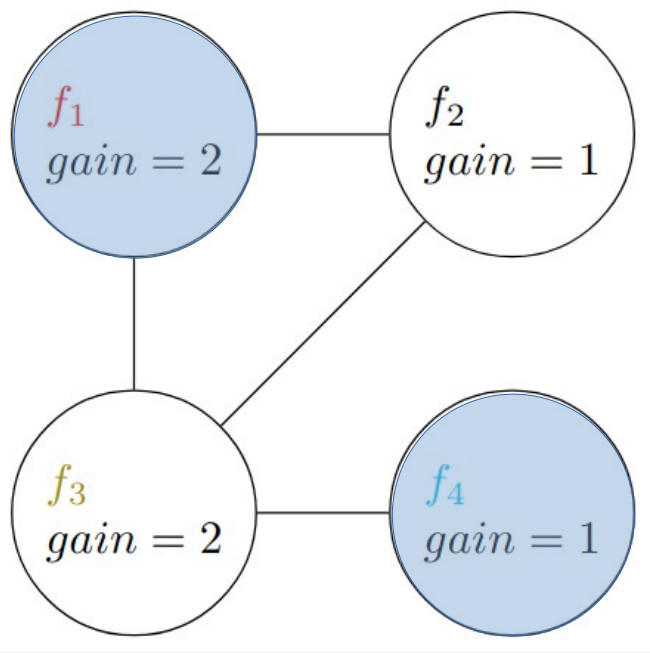
```
AND w0, w22, i2;
AND w1, w0, i3;
AND w2, w1, i4;
AND w3, w2, i5;
AND w4, i6, i7;
AND o2, w3, w4;
```

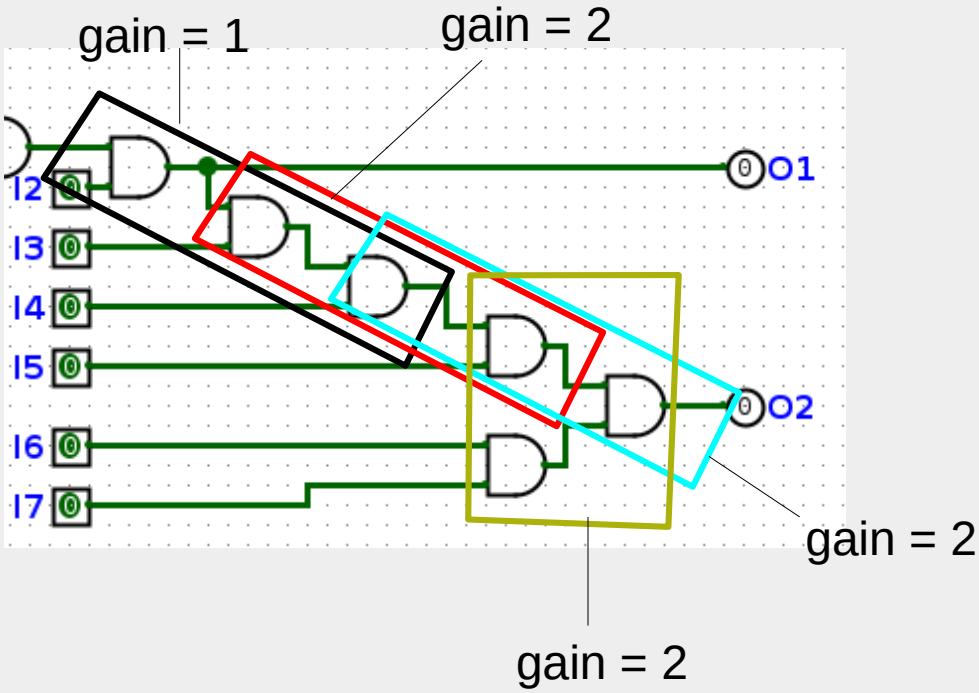




1.3 Fusion selection

```
AND w0, w22, i2;
AND w1, w0, i3;
AND w2, w1, i4;
AND w3, w2, i5;
AND w4, i6, i7;
AND o2, w3, w4;
```

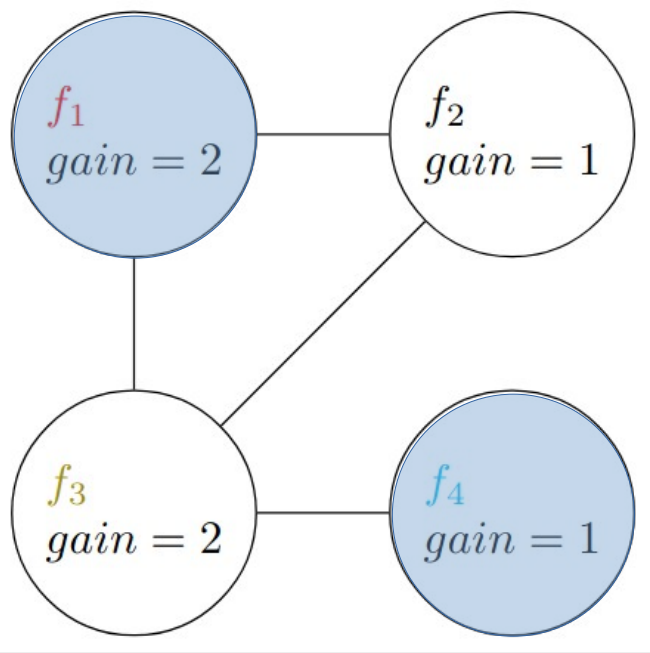


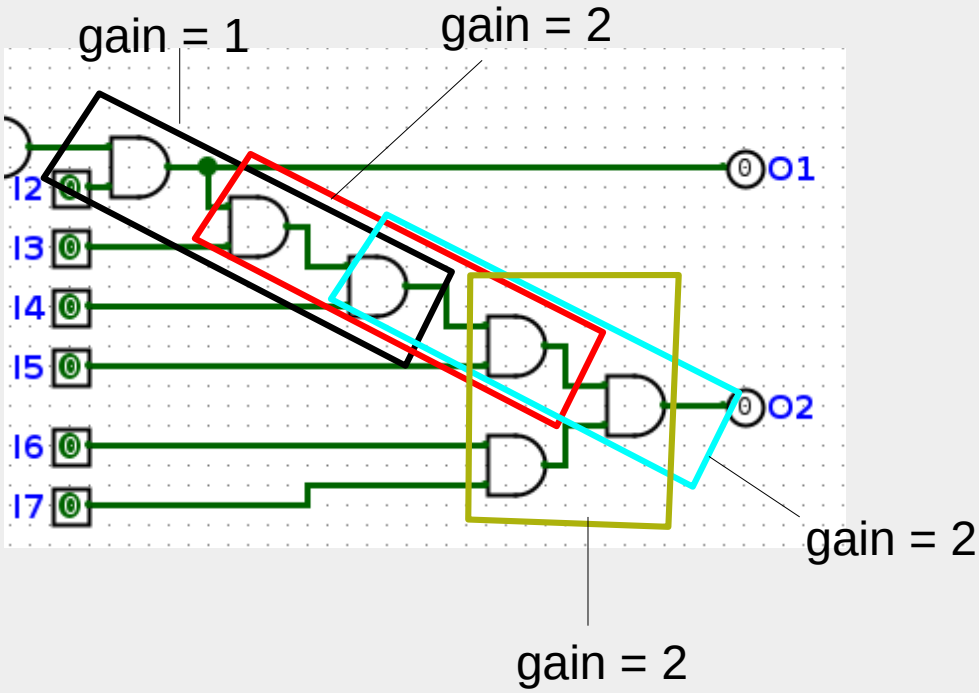


Maximum Independent Set

1.3 Fusion selection

```
AND w0, w22, i2;
AND w1, w0, i3;
AND w2, w1, i4;
AND w3, w2, i5;
AND w4, i6, i7;
AND o2, w3, w4;
```





Maximum Independent Set problem !

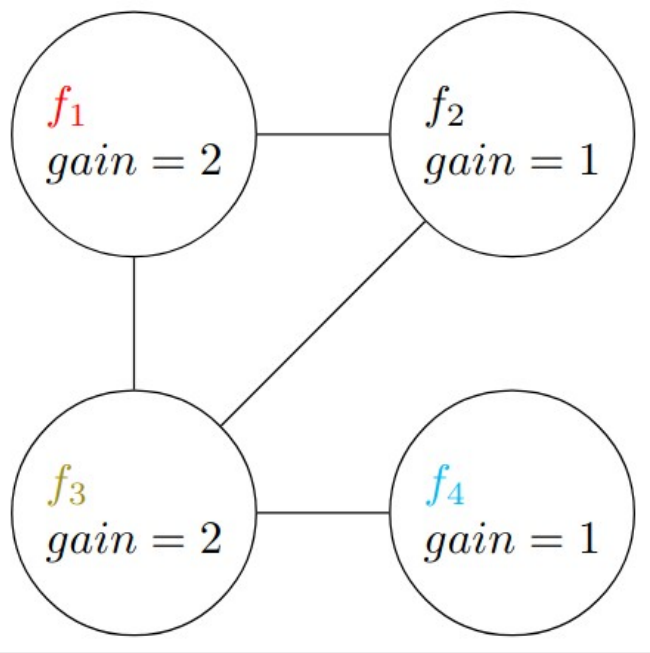
1.3 Fusion selection : Heuristic

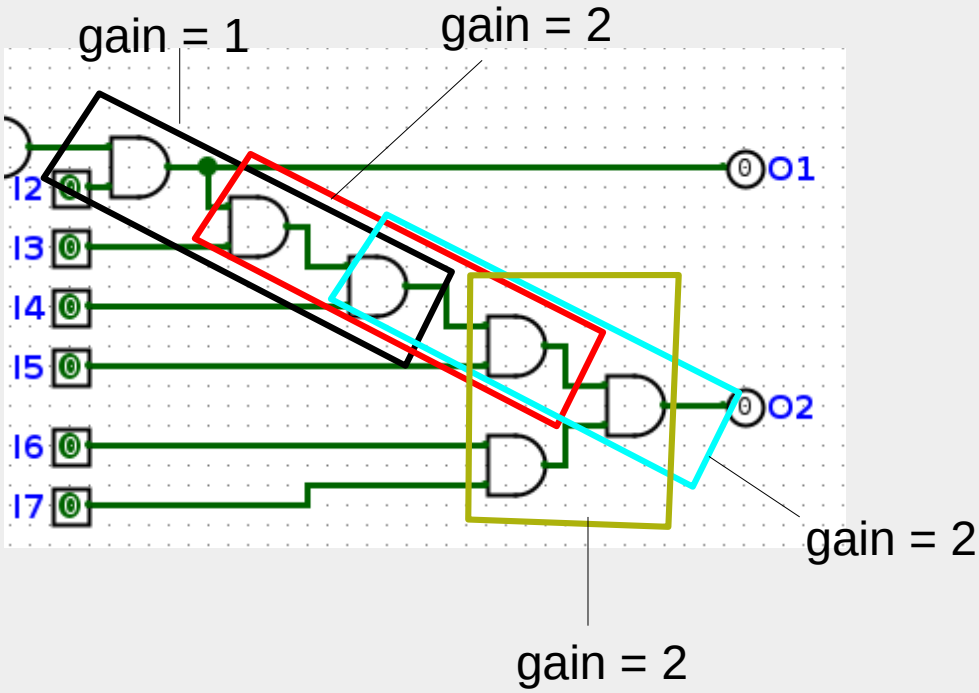
- Given a collision zone :

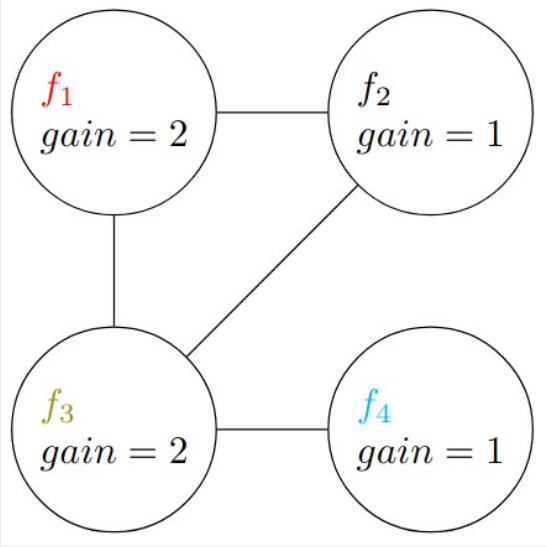
```
sort it (by degree ascending, gain descending)
while(zone.isNotEmpty):
    f = zone.pop()
    selection.add(f)
    selection.addAll(fusions excluded by f)
```

1.3 Fusion selection

```
AND w0, w22, i2;
AND w1, w0, i3;
AND w2, w1, i4;
AND w3, w2, i5;
AND w4, i6, i7;
AND o2, w3, w4;
```





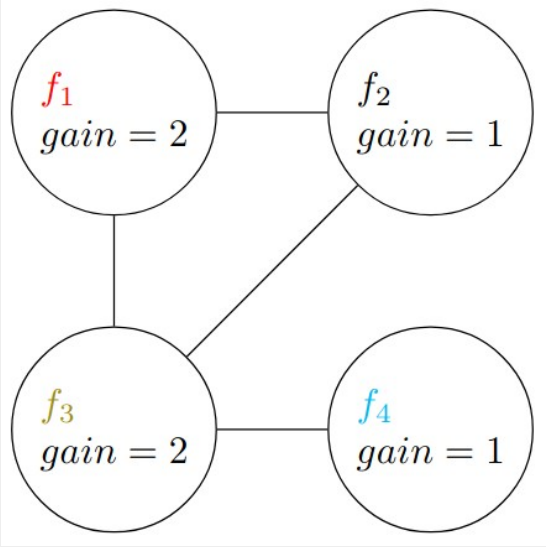


f4 (gain=1) ; f1 (gain=2) ; f2 (gain=1) ; f3 (gain=2)

degree = 1

degree = 2

degree = 3

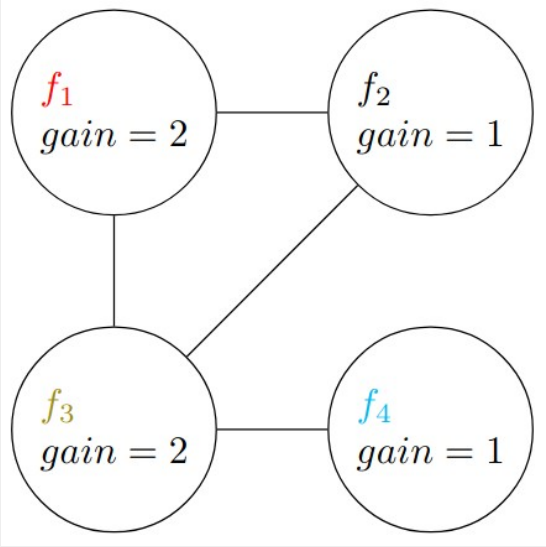


f4 (gain=1) ; f1 (gain=2) ; f2 (gain=1) ; ~~f3 (gain=2)~~

degree = 1

degree = 2

degree = 3

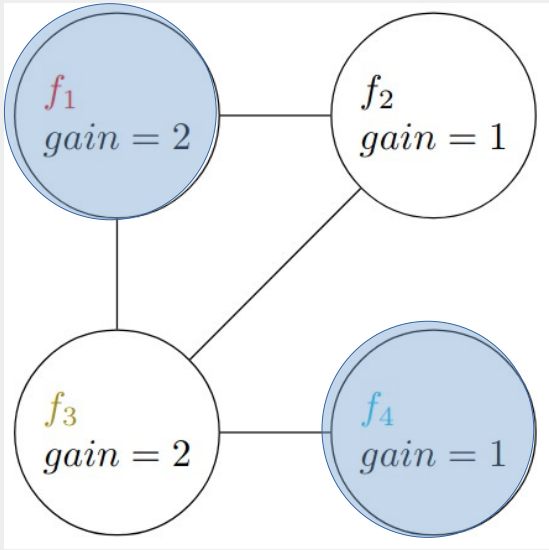


f4 (gain=1) ; f1 (gain=2) ; ~~f2 (gain=1)~~ ; ~~f3 (gain=2)~~

degree = 1

degree = 2

degree = 3

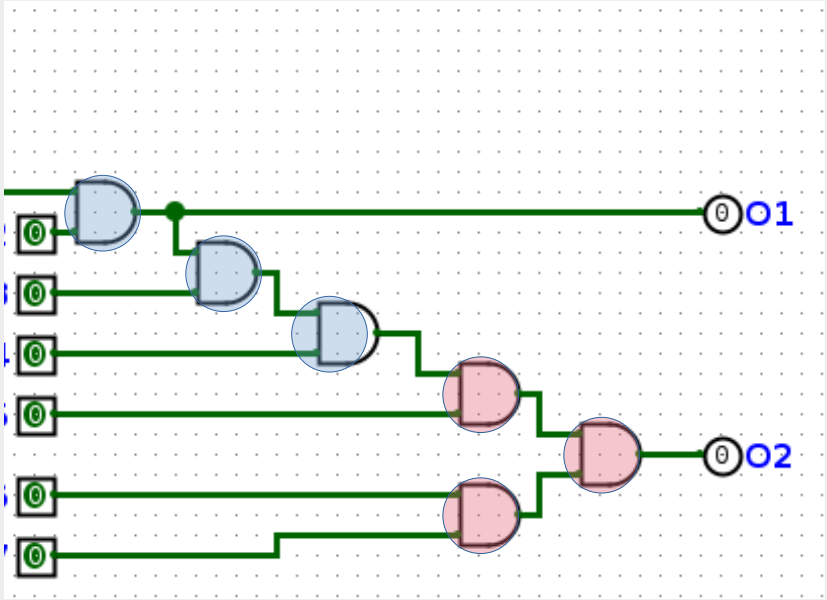


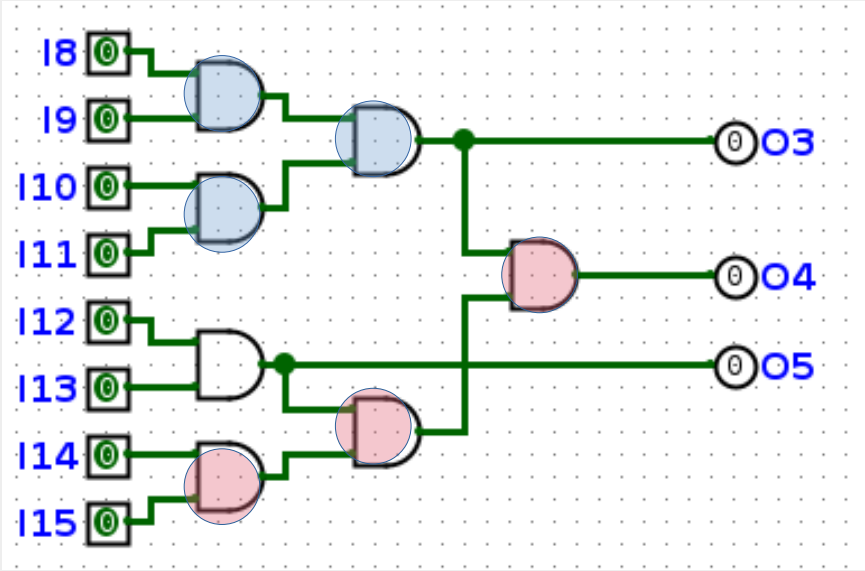
f4 (gain=1) ; f1 (gain=2) ; ~~f2 (gain=1)~~ ; ~~f3 (gain=2)~~

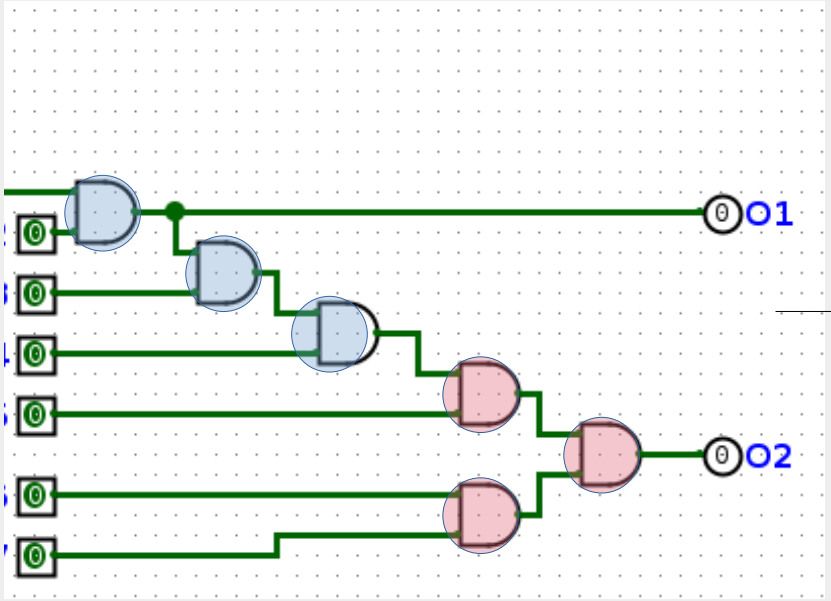
degree = 1

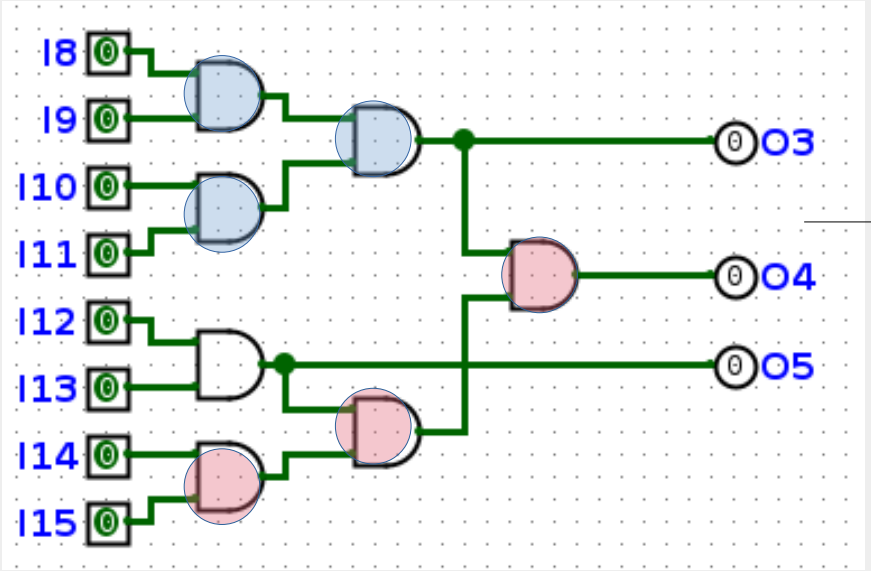
degree = 2

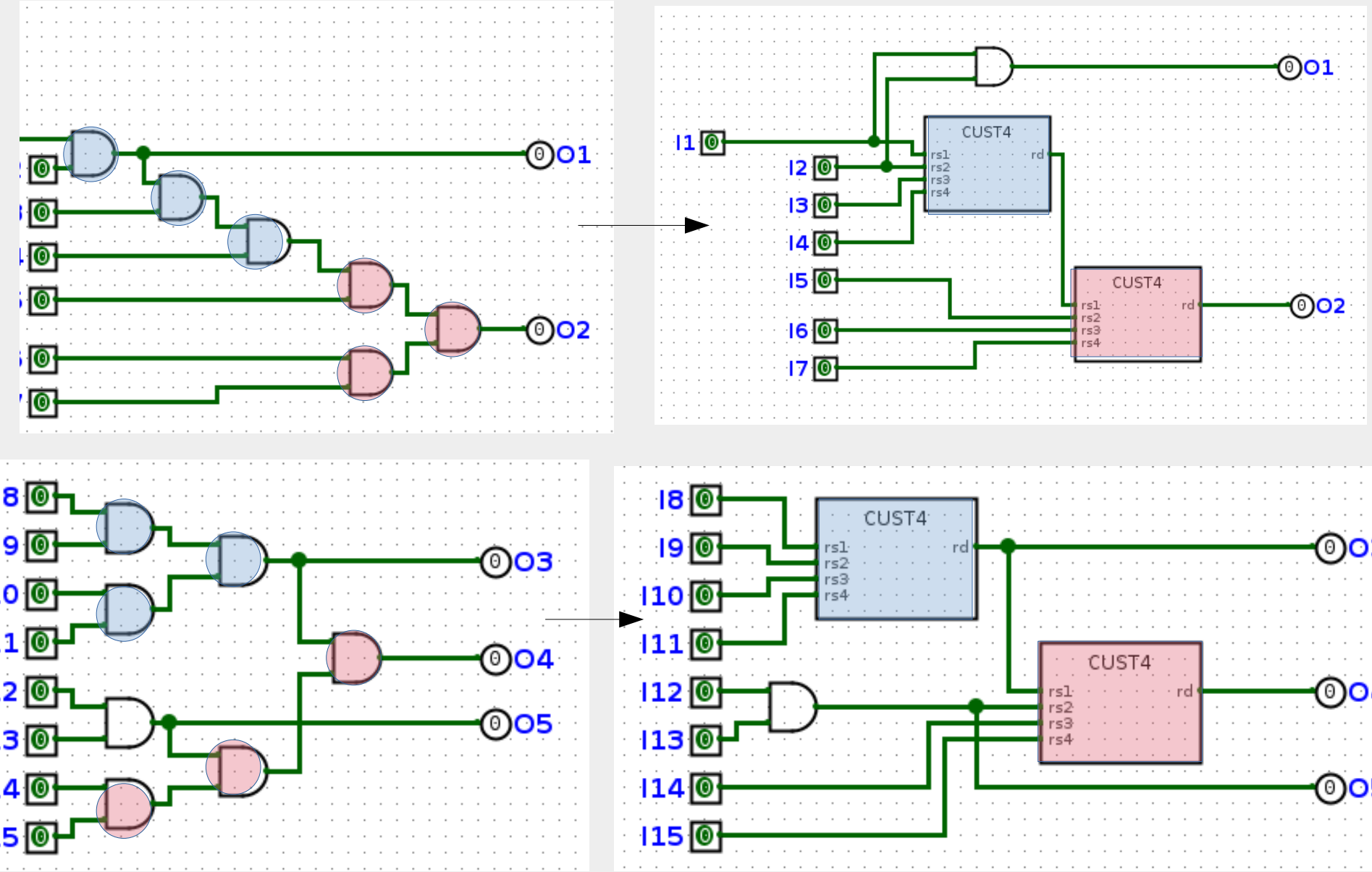
degree = 3


O1
O2

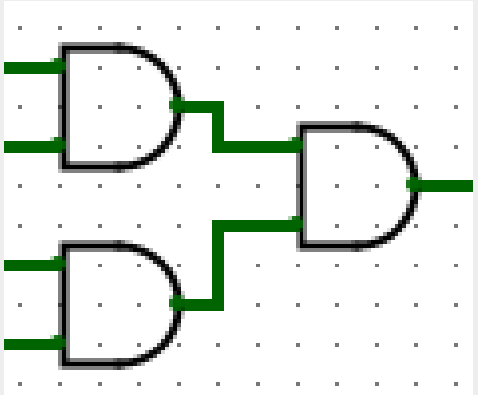

I8
I9
I10
I11
I12
I13
I14
I15
O3
O4
O5


O1
O2


I8
I9
I10
I11
I12
I13
I14
I15
O3
O4
O5

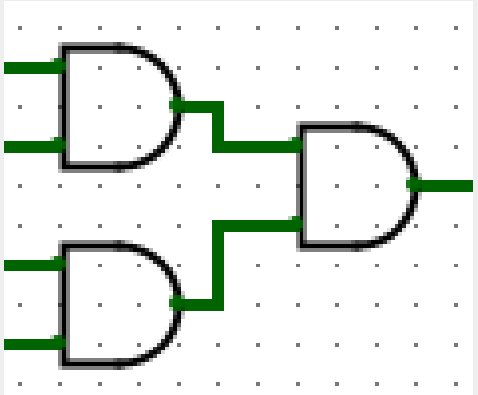

O1
O2
I1
I2
I3
I4
I5
I6
I7
CUST4
rs1
rs2
rs3
rs4
rd
I8
I9
I10
I11
I12
I13
I14
I15
O3
O4
O5

2. Merging reused operands



Towards more flexibility...

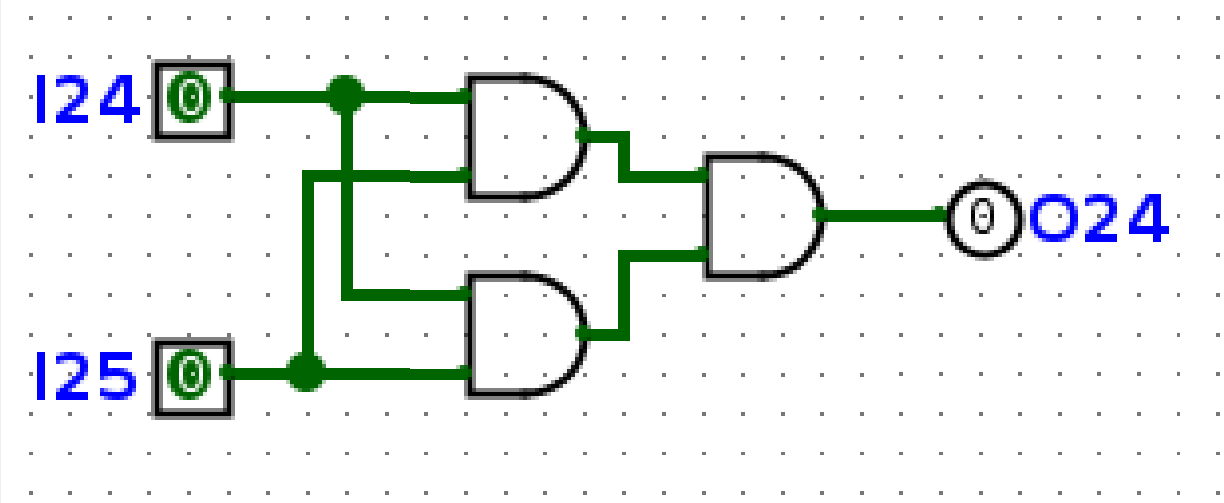
2. Merging reused operands



```
AND w20, i24, i25;
AND w21, i14, i25;
AND o24, w20, w21;
```

Towards more flexibility...

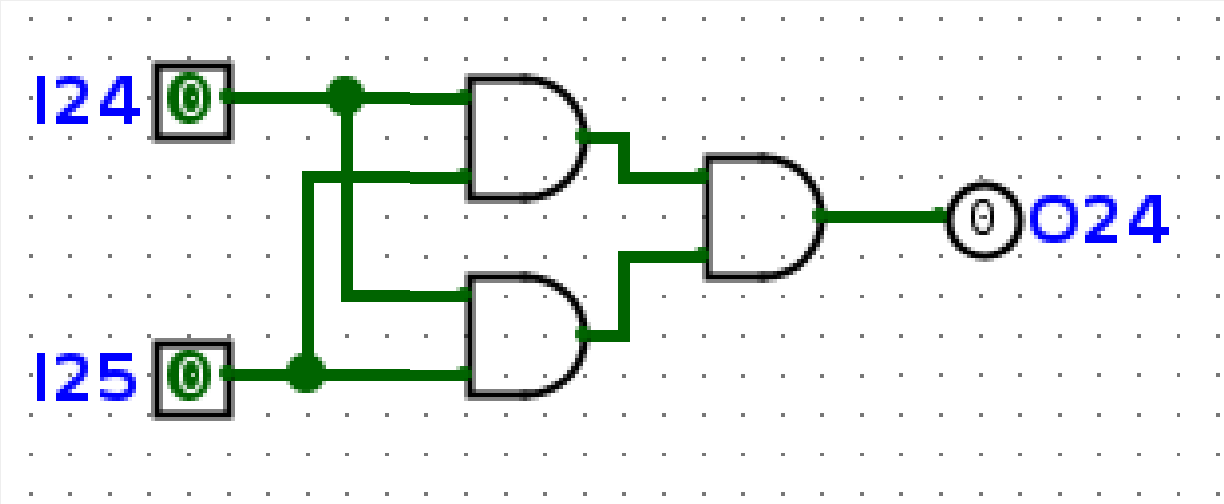
2. Merging reused operands



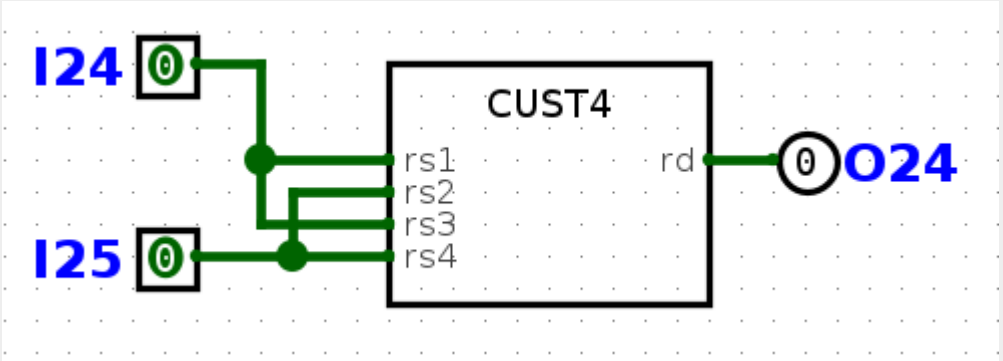

```
AND w20, i24, i25;
AND w21, i14, i25;
AND o24, w20, w21;
```

Towards more flexibility...

2. Merging reused operands

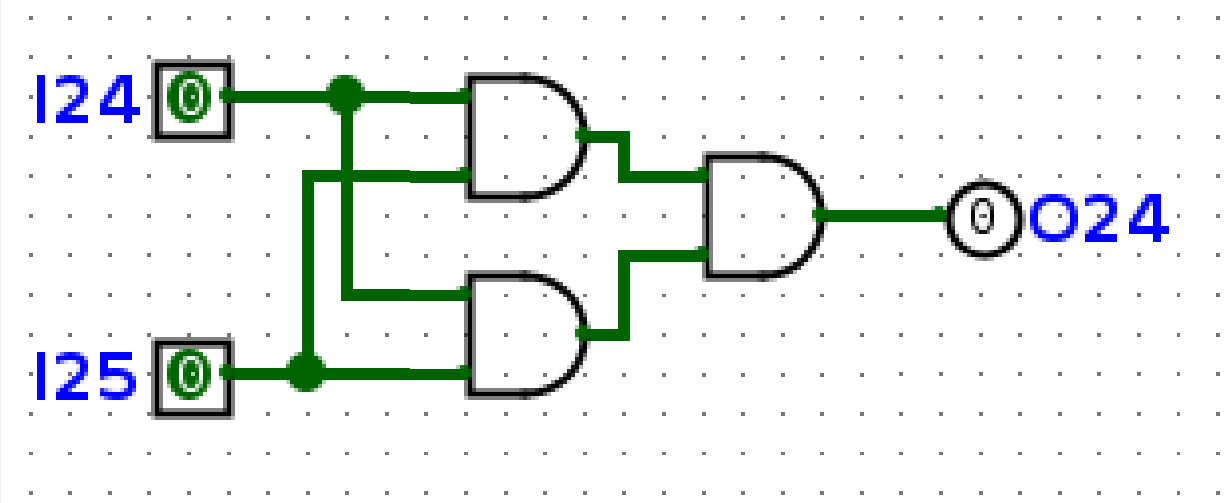


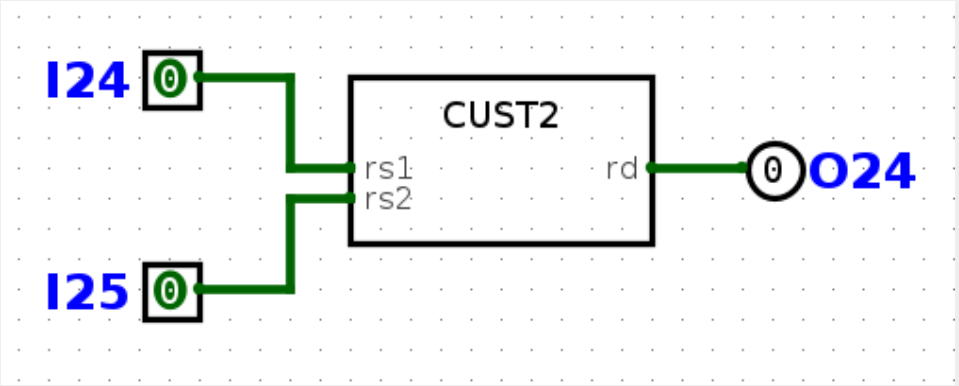

```
AND w20, i24, i25;
AND w21, i14, i25;
AND o24, w20, w21;
```

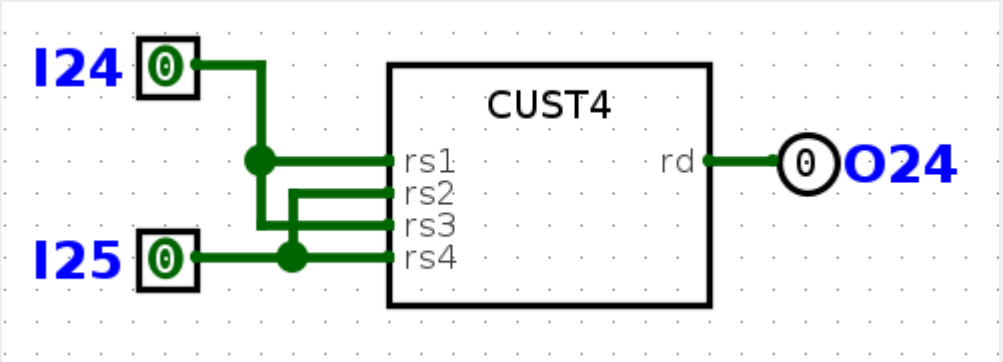


Towards more flexibility...

2. Merging reused operands





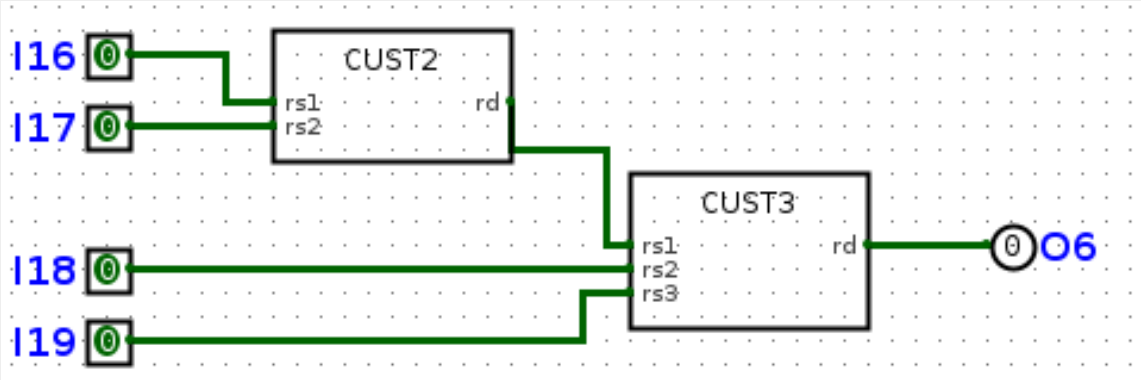


Towards more flexibility...

4-inputs fusion examples (2)

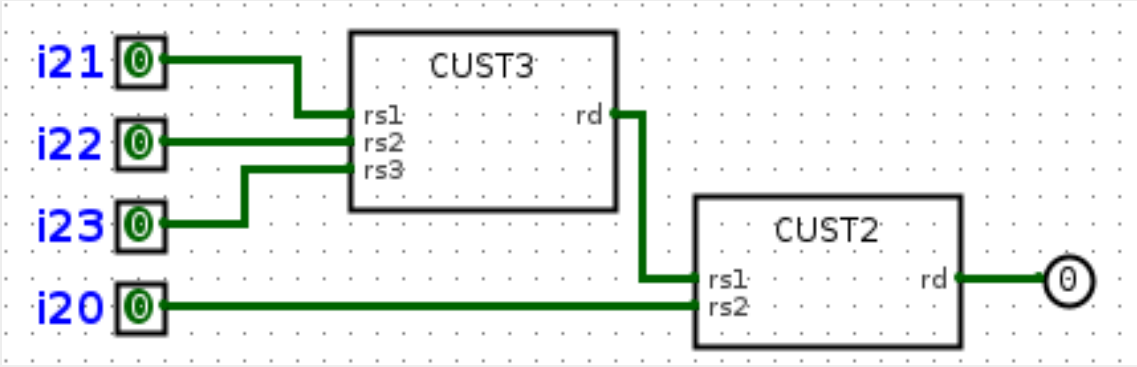
- “C-fusion”

```
CUST w13, [CUST2], i16, i17;
CUST o6, [CUST3], w13, i18, i19;
```

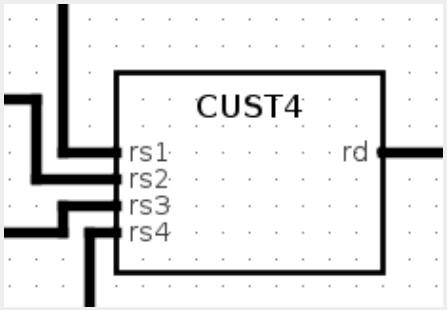


- “D-fusion”

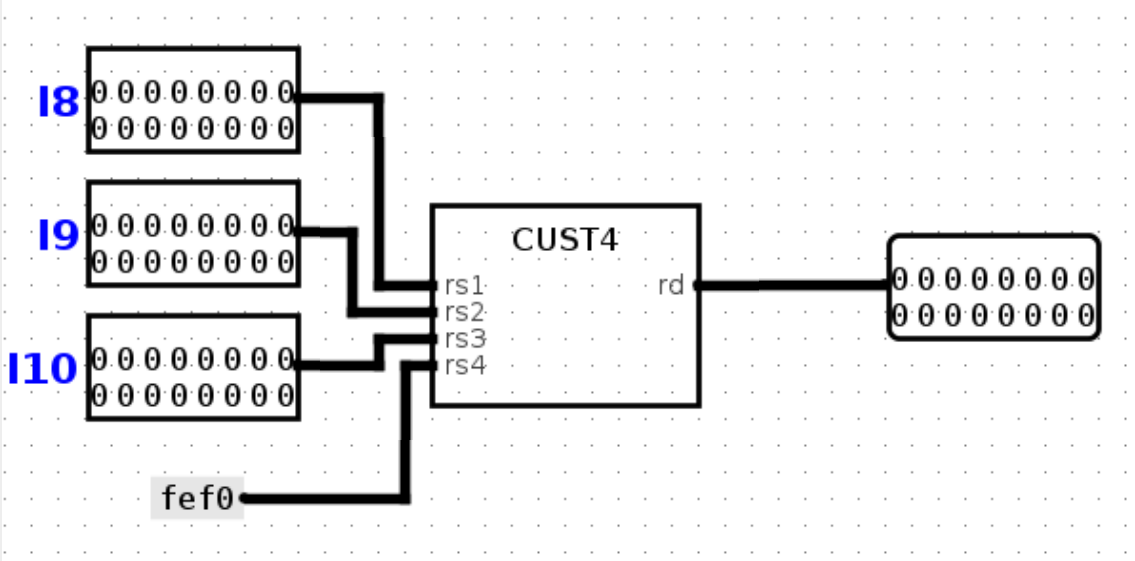
```
CUST w17, [CUST3], i21, i22, i23;
CUST o20, [CUST2], w17, i20;
```



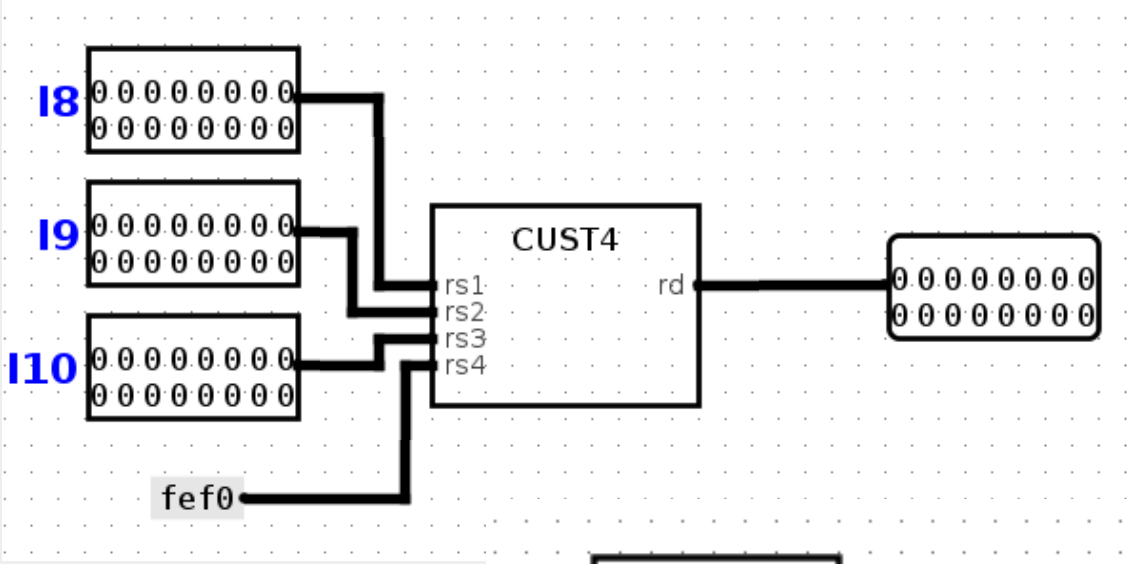
3. Arity reduction : Constant folding

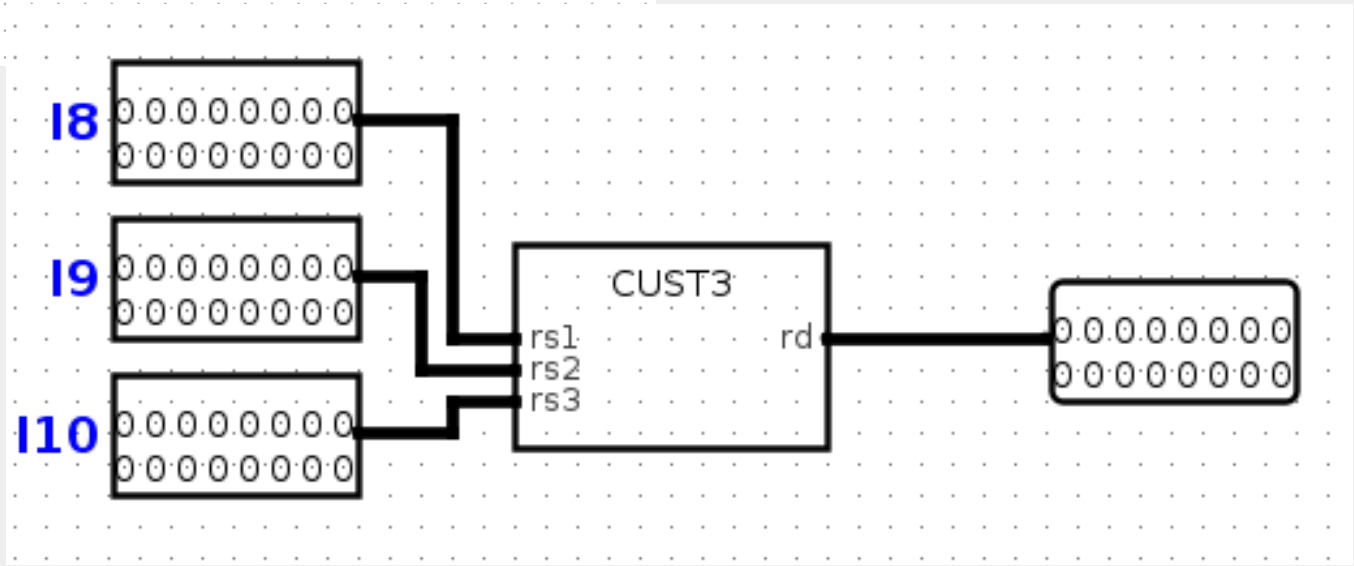


3. Arity reduction : Constant folding



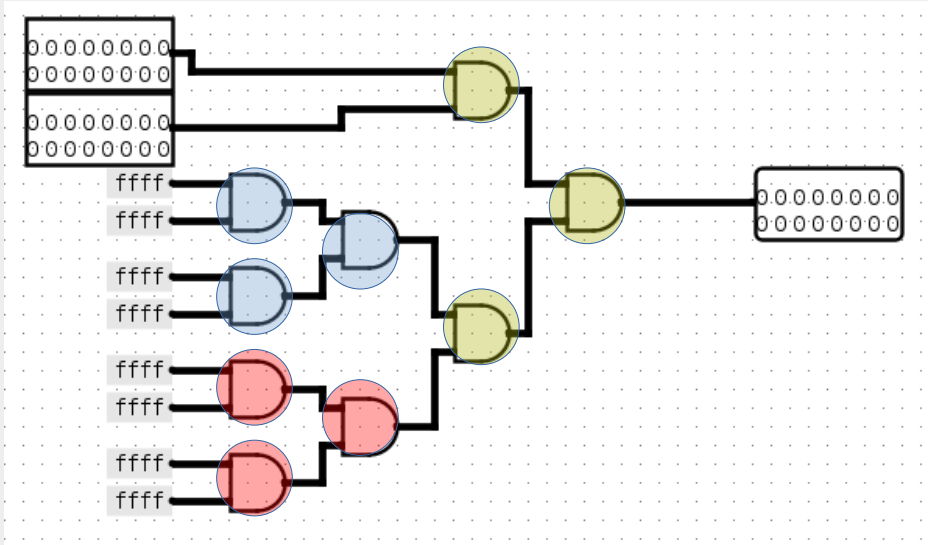
3. Arity reduction : Constant folding



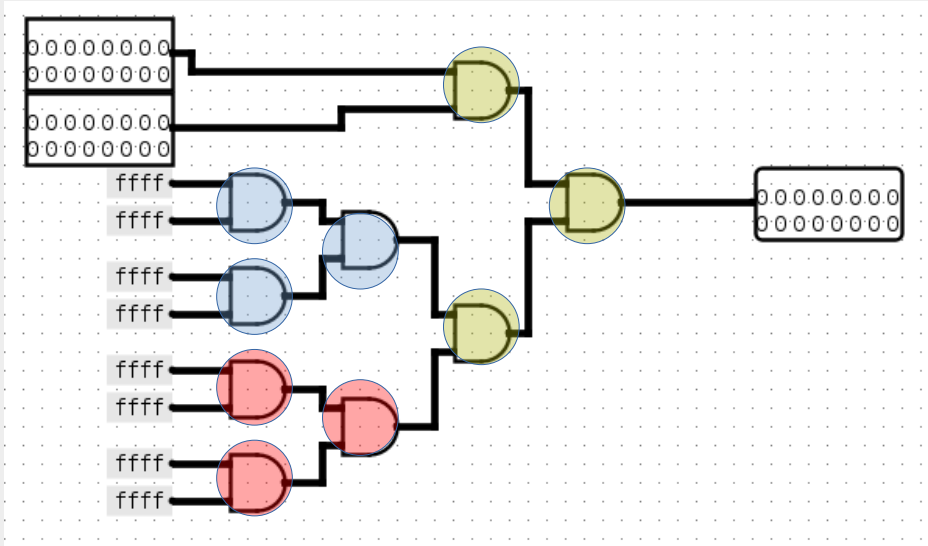


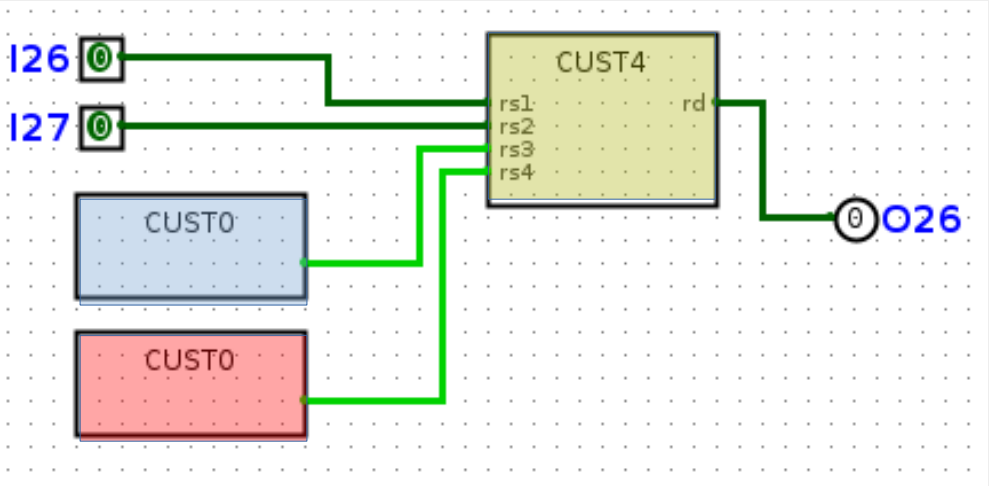
4. Merging nodes of arity 0

4. Merging nodes of arity 0

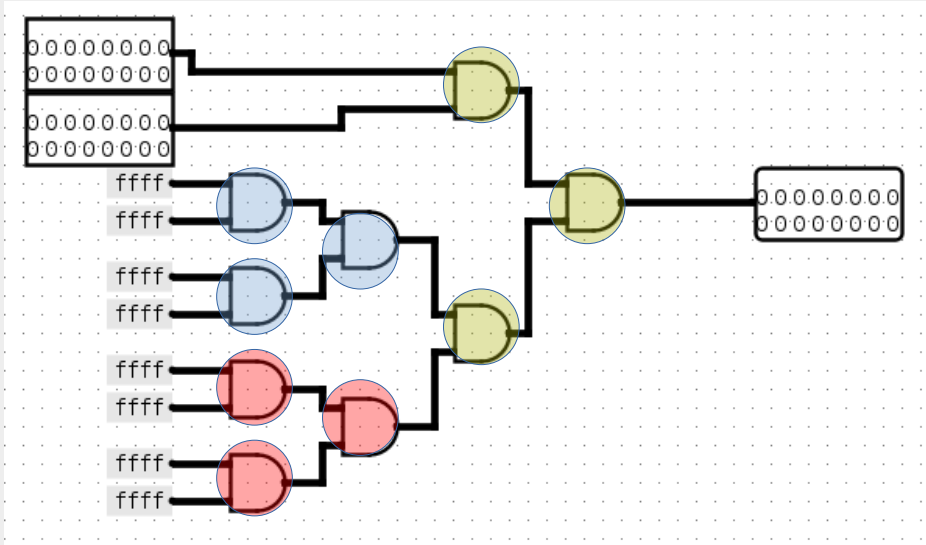


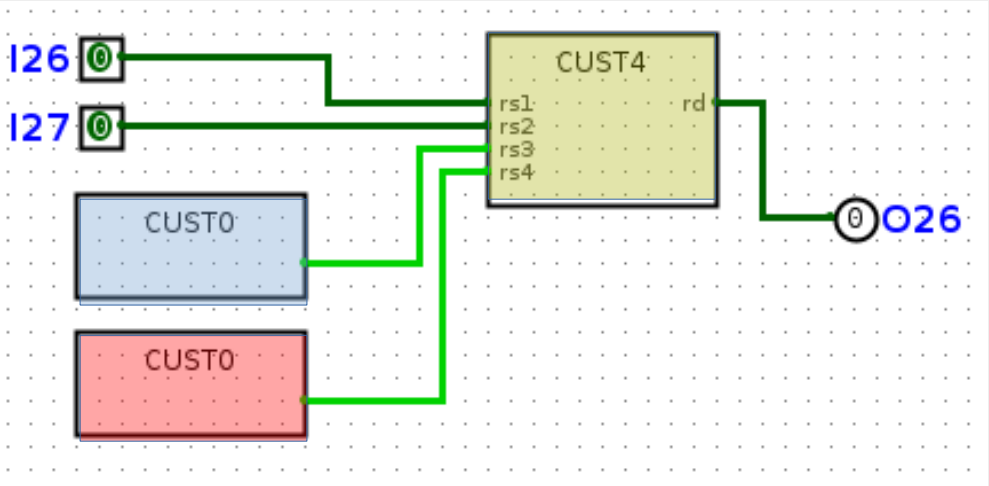
4. Merging nodes of arity 0

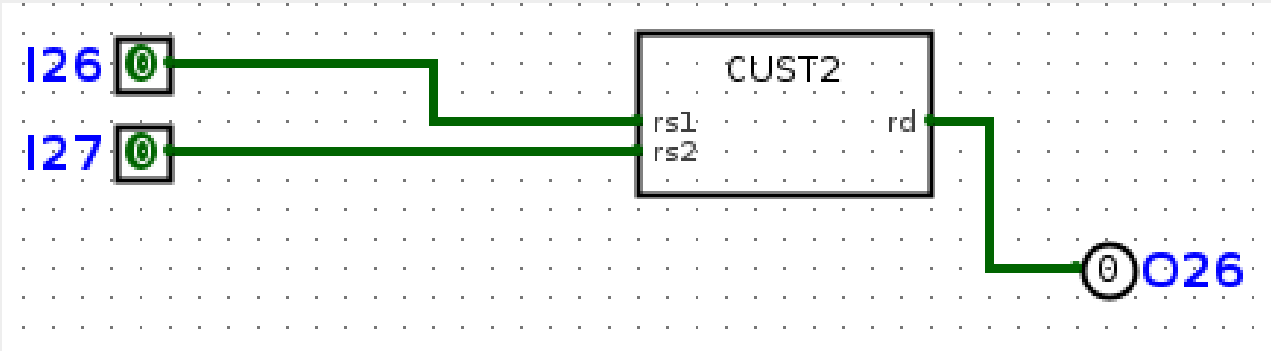




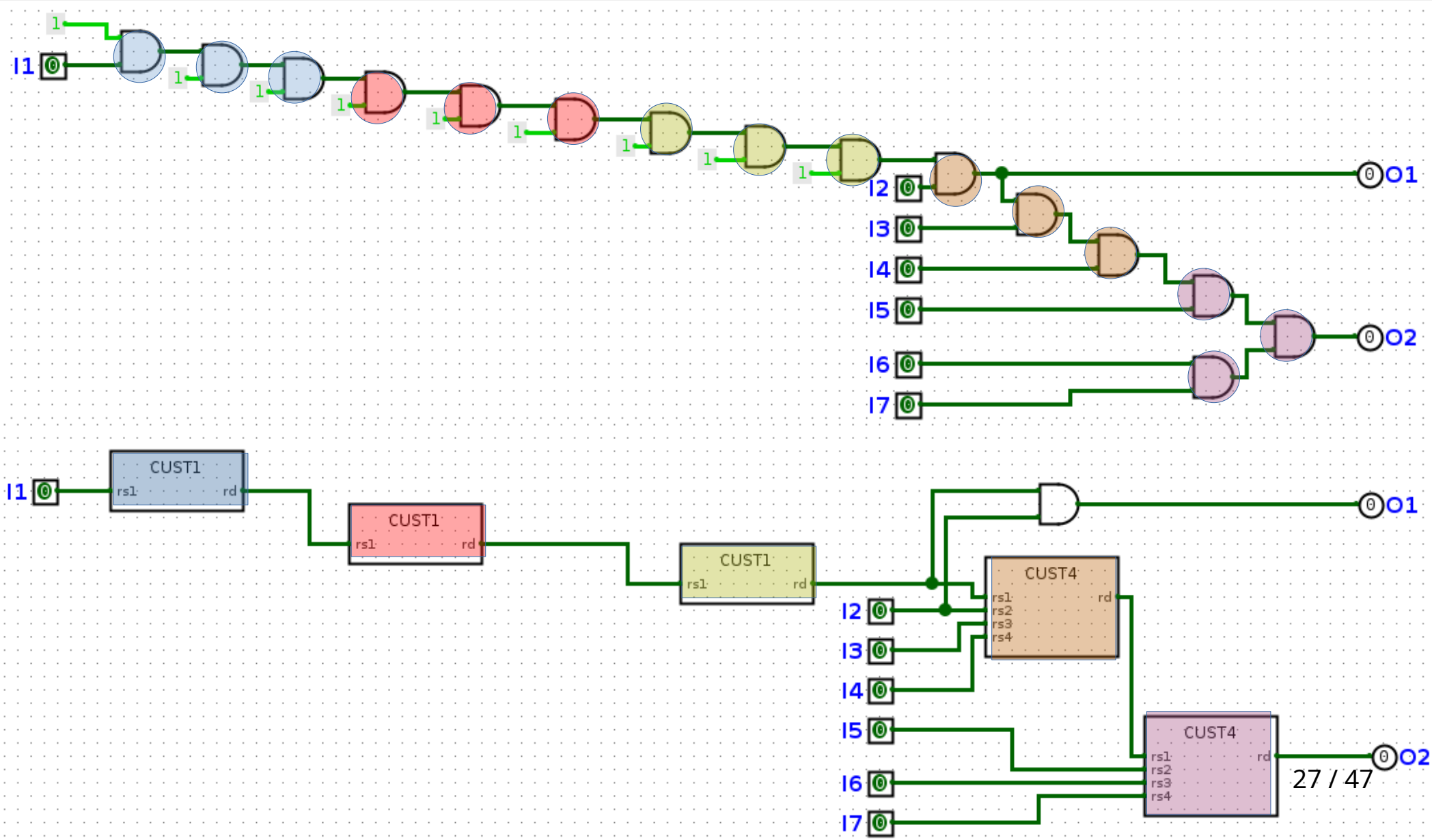
4. Merging nodes of arity 0



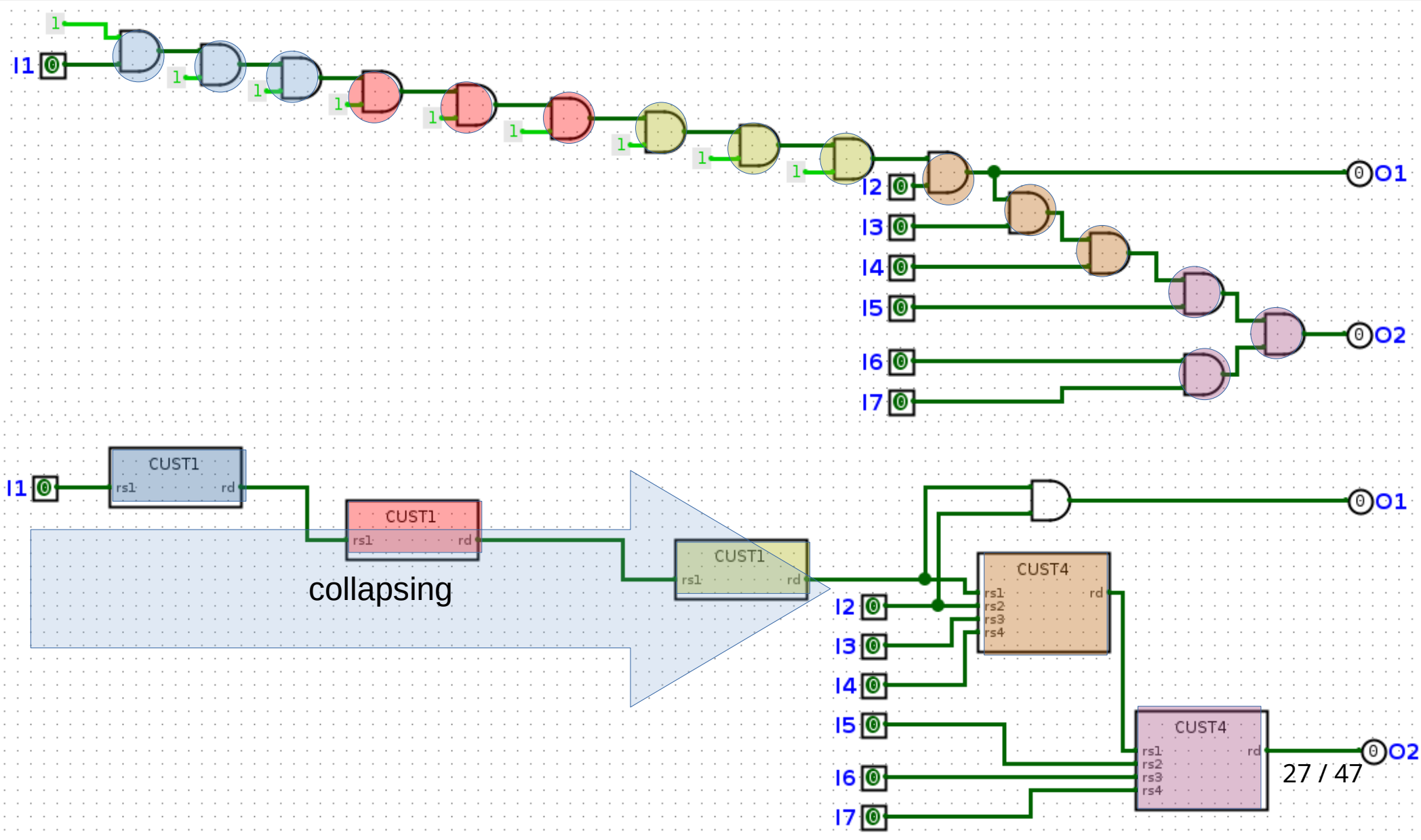


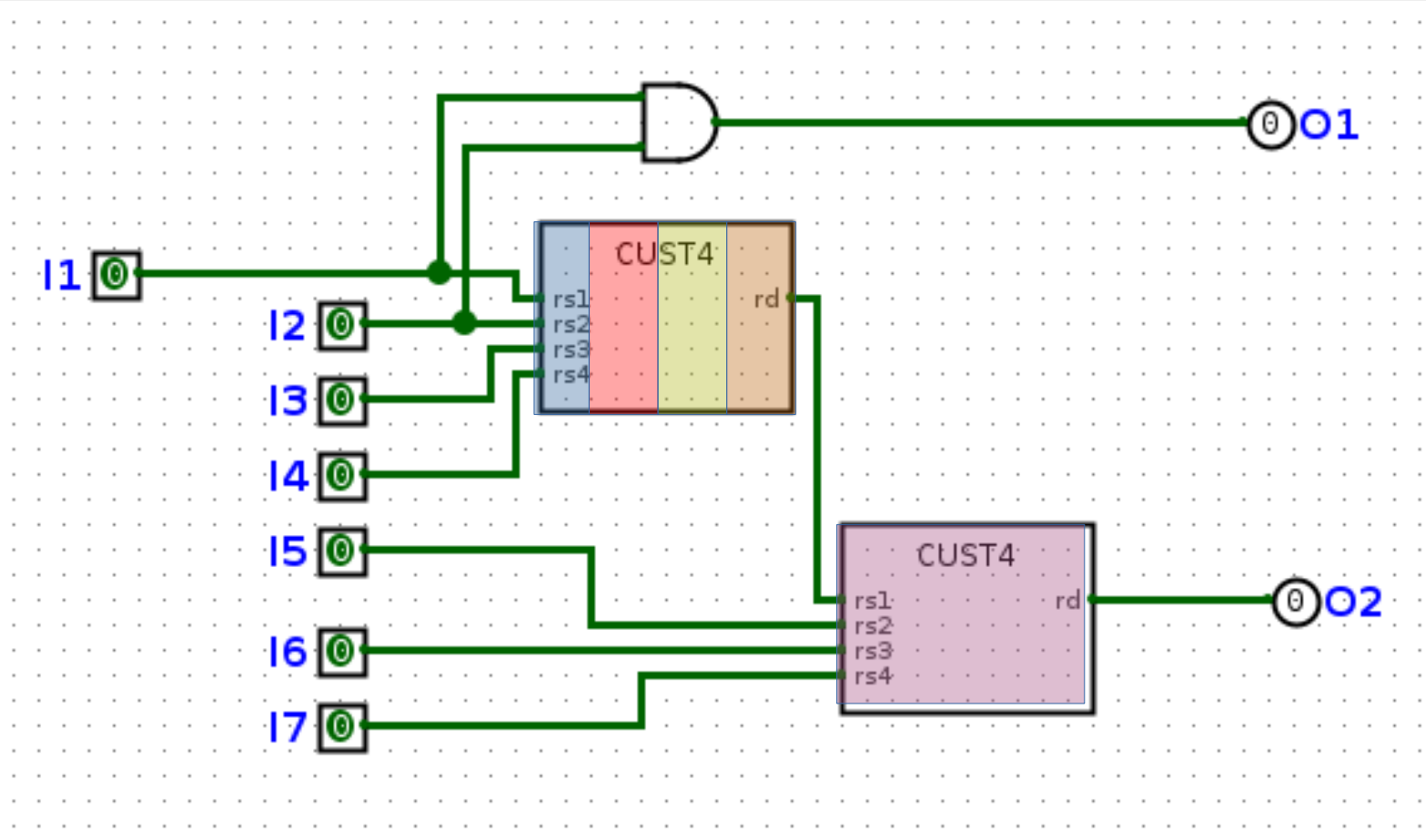


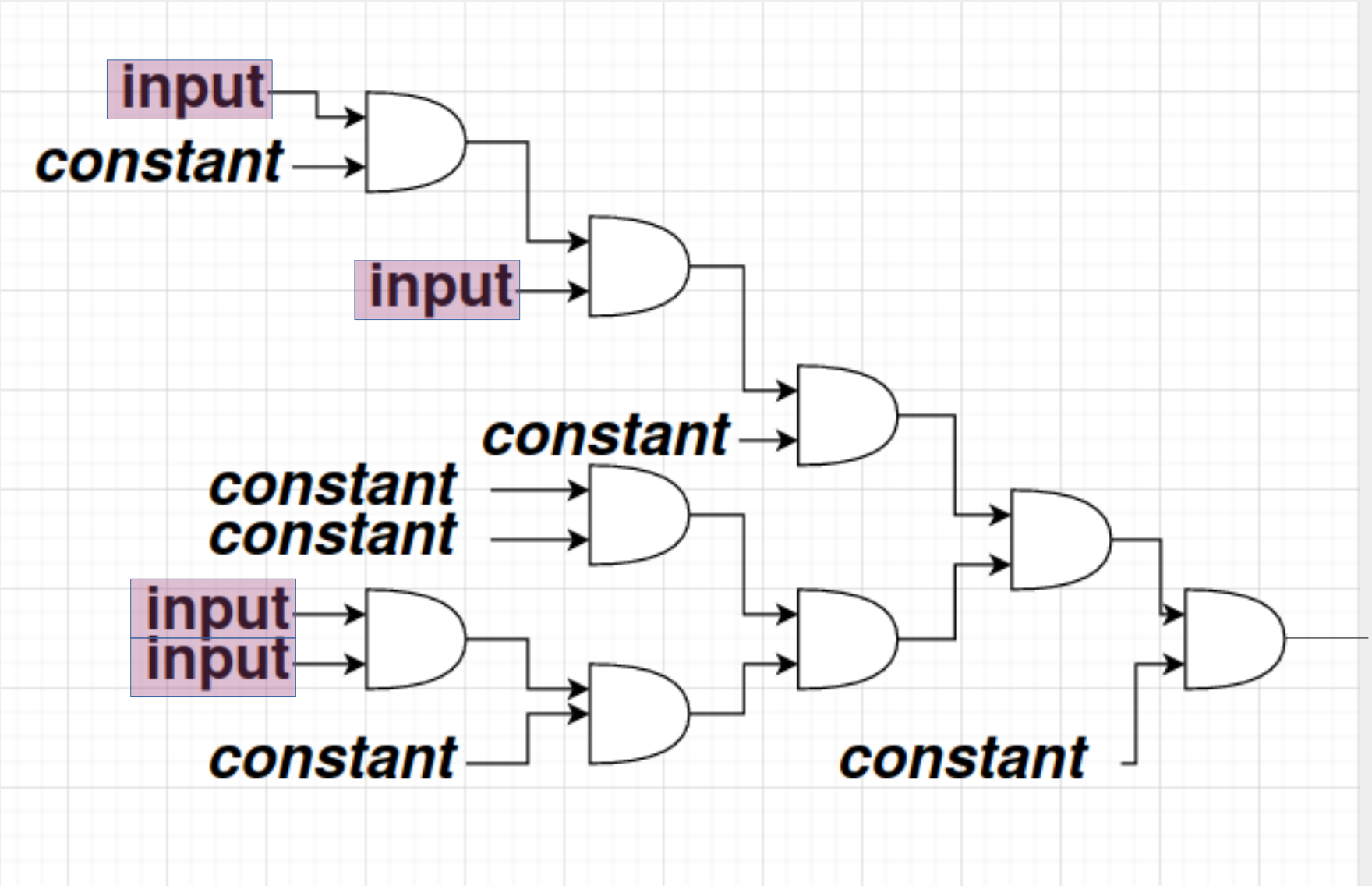
5. Collapsing chains of arity-1 nodes



5. Collapsing chains of arity-1 nodes




O1
I1
CUST4
rs1
rs2
rs3
rs4
rd
I2
I3
I4
I5
I6
I7
CUST4
rs1
rs2
rs3
rs4
rd
O2


input
constant
input
constant
constant
constant
input
input
constant
constant

6. Repeat

- At start, only arity-2 nodes
- Now we have all kind of different nodes
- Rerun to find new fusions

Limit the number of fusions

- In HW, bounded to 32 distinct custom functions

$$Manticore\ functions \subseteq \{ \mathbb{N}_{16}^{4} \rightarrow \mathbb{N}_{16} \}$$

- One function can correspond to many fusions

Problems (1)

- Best fusions at an iteration may not be the best over iterations
- Useless constant merging
- Performance : Exponential runtime for optimal choice of fusions

Problems (2)

- Function equality

$$(((a \& b) \& c) \& d) == ((a \& b) \& (c \& d))$$

$$((a \& b) \& (a \& b)) == (a \& b)$$

$$((a \mid b) \wedge (c \& d)) == ((c \& d) \wedge (a \mid b))$$

Benchmarks

- Mips32 : a 32 bits MIPS processor
- PicoRV32 : small 32 bits RISC-V processor
- Swizzle : inversion of a simple bit vector
- XorReduce : XOR bit reduction of bit vector
- Xormix32 : pseudorandom number generator
- Bitcoin : a bitcoin mining simulation

Benchmarks : additional information

	#collision zones	largest collision zone
Mips32	4	15
PicoRV32	72	48
Swizzle	1	17
XorReduce	1	37
Xormix32	36	291
Bitcoin	12	11

Benchmark results

- Random choice

	#Cycles before	#Cycles after	Speedup	Custom Functions	Fusions	Loop iterations
MIPS32	335	315	5.97%	13	23	3
PicoRV32	2028	1767	12.87%	32	216	2
Swizzle	57	46	19.30%	4	4	3
XorReduce	107	87	18.69%	4	14	3
Xormix32	1808	1269	29.81%	32	431	2
Bitcoin	266	225	15.41%	14	141	4

Benchmark results

- Smarter choice

	#Cycles before	#Cycles after	Speedup	Custom Functions	Fusions	Loop iterations
MIPS32	335	315	5.97%	13	23	2
PicoRV32	2028	1770	12.72%	32	175	2
Swizzle	57	46	19.30%	4	4	3
XorReduce	107	85	20.56%	2	12	2
Xormix32	1808	1092	39.60%	32	384	2
Bitcoin	327	279	14.68%	13	119	3

Benchmarks : Unbounded functions

- HW with support for infinitely many LUTs
- Random choice

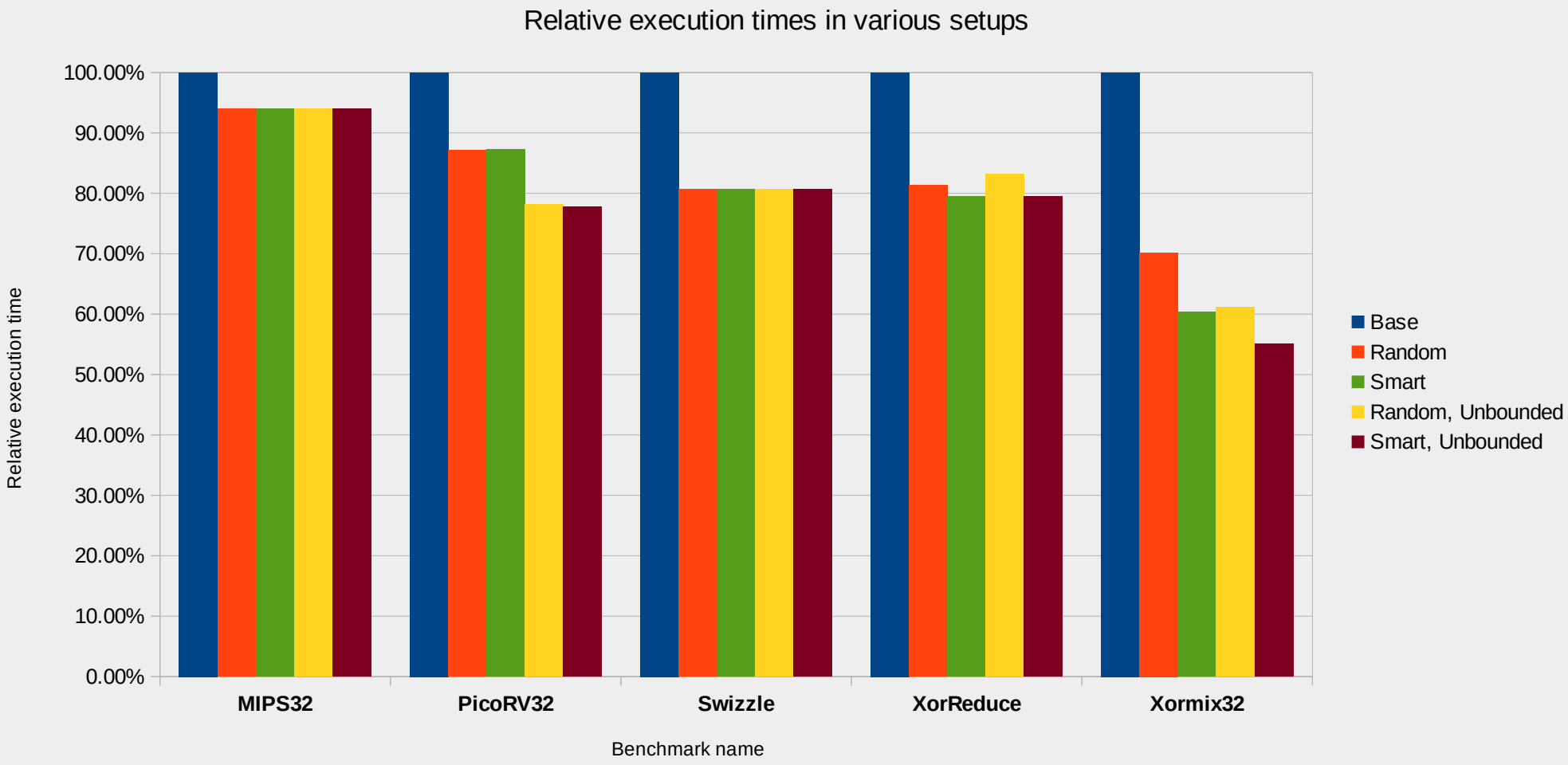
	#Cycles before	#Cycles after	Speedup	Custom Functions	Fusions	Loop iterations
MIPS32	335	315	5.97%	13	23	2
PicoRV32	2028	1528	21.84%	64	319	3
Swizzle	57	46	19.30%	4	4	3
XorReduce	107	89	16.82%	2	10	2
Xormix32	1808	1105	38.88%	44	360	4
Bitcoin	219	187	14.61%	13	117	3

Benchmark results

- HW with support for infinitely many LUTs
- Smarter choice

	#Cycles before	#Cycles after	Speedup	Custom Functions	Fusions	Loop iterations
MIPS32	335	315	5.97%	13	23	2
PicoRV32	2028	1578	22.19%	62	316	3
Swizzle	57	46	19.30%	4	4	3
XorReduce	107	85	20.56%	2	12	2
Xormix32	1808	995	44.97%	32	364	3
Bitcoin	211	182	13.74%	14	120	3

Benchmarks results



Conclusion

- Manticore : Parallelize RTL simulation on FPGAs
- Compiler : Pass to find custom functions
- Algorithm : Fixed-point based loop
  - Pattern matching search
  - Selection (random/locally optimal/heuristic)
  - Transformation (arity reduction, collapsing)
- ~20% speedup on some circuits